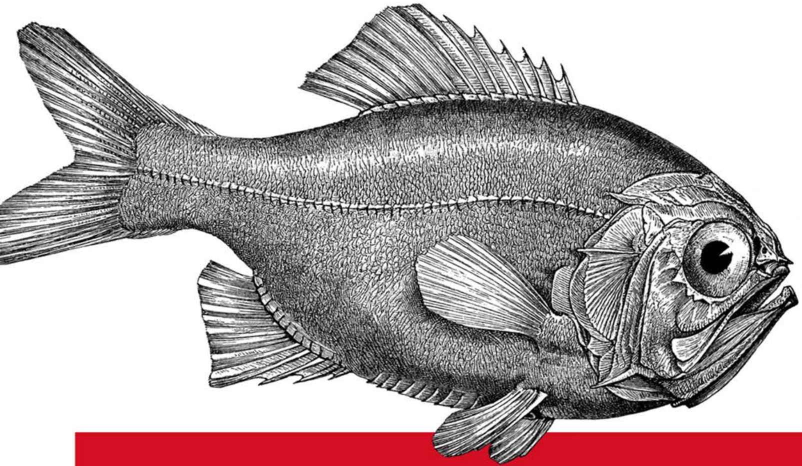
O'REILLY®

图灵程序设计丛书



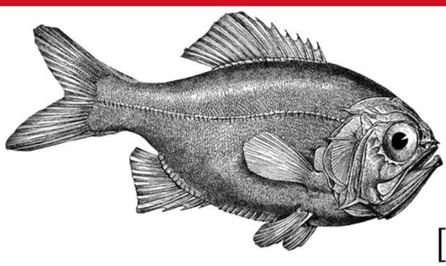
去中心化应用

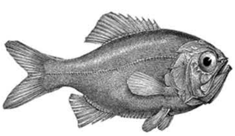
区块链技术概述

Decentralized Applications

帮助读者全面理解并创建dapp

实现超越Web应用的安全性、隐私性、灵活性





[美] Siraj Raval 著

吴海星 译

中国工信出版集团　人民邮电出版社
POSTS & TELECOM PRESS

图书介绍

《区块链技术进阶与实战》

- 专注介绍区块链核心原理和应用技术
- 详细解读区块链平台以太坊和HyperLedger
- 注重实战，全书包含5个完整实际项目案例

作者：蔡亮 李启雷 梁秀波

《你不知道的JavaScript（上、中、下卷）》

- 深入挖掘JavaScript语言本质，简练形象地解释抽象概念，打通JavaScript的任督二脉
- 探索JavaScript语言核心概念
- 深入了解ES6，展望JavaScript发展方向

作者：Kyle Simpson

《Go并发编程实战（第2版）》

- 国内知名Go语言技术布道者的再造之作
- 李响、谢孟军、刘奇、左玥、肖德时、Googol Lee、田琪等业内大咖倾心力捧
- 深入Go语言及其并发原理，挖出并发编程最佳实践，百样示例助你成为多核计算时代的Go程序高手

作者：郝林

数字版权声明

TURING 图灵程序设计丛书

去中心化应用：区块链技术概述

Decentralized Applications:
Harnessing Bitcoin's Blockchain Technology

[美] Siraj Raval 著
吴海星 译

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北京

图书在版编目（CIP）数据

去中心化应用 : 区块链技术概述 / (美) 西拉杰•拉瓦尔 (Siraj Raval) 著 ; 吴海星译. -- 北京 : 人民邮电出版社, 2018.5
（图灵程序设计丛书）
ISBN 978-7-115-47930-3

Ⅰ. ①去… Ⅱ. ①西… ②吴… Ⅲ. ①电子商务一支付方式 Ⅳ. ①F713.361.3

中国版本图书馆CIP数据核字(2018)第032942号

内 容 提 要

在这本实用指南中，作者解释了为什么去中心化应用（dapp）将比现在流行的 Web 应用得到更广泛的使用以及实现更多盈利，展示了如何使用现有工具来创建可用的 dapp 及其市场，并研究了目前两个成功的 dapp 案例。读者将了解到区块链的加密存储台账、scarce-asset 模型和点对点技术如何提供比当前软件模型更灵活、更具激励性的结构。

本书读者对象为对区块链感兴趣的开发人员。

◆ 著　　[美] Siraj Raval
译　　吴海星
责任编辑　杨　琳
责任印制　周昇亮

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　印刷

◆ 开本：880×1230　1/32
印张：3.75
字数：123千字　　2018年 5 月第 1 版
印数：1 – 3 500册　　2018年 5 月北京第 1 次印刷

著作权合同登记号　图字：01-2017-9358号

定价：39.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字 20170147 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

目录

前言

排版约定

本书使用下列排版约定。

- 等宽字体（`constant width`）
  表示广义上的计算机编码，包括变量或函数名、数据库、数据类型、环境变量、语句和关键字。

- 等宽粗体（**`constant width bold`**）
  表示应该由用户按照字面输入的命令或其他文本。

- 等宽斜体（*`constant width italic`*）
  表示应该由用户替换或取决于上下文的值。

代码示例

补充材料（包括代码示例、练习题等）可以从 https://github.com/oreillymedia/decentralized_applications 下载。

本书旨在帮助你做好工作。一般来说，你可以在程序和文档中使用本书的代码。除非你使用了很大一部分代码，否则无须联系我们获取许可。例如，使用来自本书的几段代码编写一个程序不需要许可。销售和分发 O'Reilly 书中用例的光盘需要许可。通过引用本书用例和代码来回答问题不需要许

可。把本书中的大量用例代码并入你的产品文档需要许可。

我们很希望但不强求注明信息来源。一条信息来源通常包括书名、作者、出版社和 ISBN。例如：“*Decentralized Applications* by Siraj Raval (O’Reilly). Copyright 2016 Siraj Raval, 978-1-4919-2454-9”。

如果你感到对示例代码的使用超出了正当引用或者这里给出的许可范围，请随时通过 permissions@oreilly.com 联系我们。

Safari®在线图书

Safari Books Online（http://www.safaribooksonline.com）是应运而生的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O’Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O'Reilly 的每一本书都有专属网页，你可以在那里找到本书的相关信息，包括勘误表、示例以及其他信息。本书的网站地址是：

http://shop.oreilly.com/product/0636920039334.do

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：http://facebook.com/oreilly

请关注我们的 Twitter 动态：http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：http://www.youtube.com/oreillymedia

电子书

扫描如下二维码，即可购买本书电子版。

第 1 章

什么是去中心化应用

有一种用于构建可伸缩、盈利性大型应用的新模型已经崭露头角。比特币以其加密存储台账、稀缺资产模型和对等网络技术开辟了一条新路，为这种称作**去中心化应用**（decentralized application，简称为 dapp）的新型软件提供了构建基础。虽然 dapp 刚刚得到媒体的关注，但我相信，终有一天它的应用范围会变得更加广泛，并将远远超过目前最流行的 Web 应用。它更灵活、更透明、更分散、更有弹性。与当前的软件模型相比，它的结构有更好的激励性。如果你想了解并亲自创建这样的应用，这是第一本能为你提供帮助的书。

1.1　预备知识：什么是比特币

在深入介绍 dapp 之前，我们先来聊聊比特币[1]和 Web。在过去 10 年间，我们亲眼目睹了 Web 以数量级计的急剧增长。随着与互联网连接的设备逐渐遍及全球，互联网用户的数量达到了数十亿。乍一看，互联网协议套件有着良好的通信标准：链路层将一些数据放在电线上；网络层对数据进行路由；传输层将数据持久化；应用层以应用的形式提供数据抽象。这 4 个协

注 1：在中国，比特币、以太币等“虚拟货币”不具有与货币等同的法律地位，不能在市场上流通使用。本书内容仅代表作者个人观点。——编者注

议层在数据交换上的合作天衣无缝，但可惜它们交换的不是价值。对于价值交换而言，比特币充当了这 4 层之上的第 5 个协议层。

我们现在确实已经有了在 Web 上进行支付的方法，但问题是，它们无一例外地跟效率低下的遗留系统搅在一起，比如在互联网出现之前设计的自动清算所系统（automated clearing house，ACH）。这些传统的支付系统需要依赖集中式的清算系统，因此慢得让人难以忍受。机器不应该为了清算一笔支付等上好几天。它们在持续不断地相互通信，应该有能力将数十亿的小额支付发送给对方，以计量电力和存储空间等资源，并且无须负担高额的中间商交易费用。比特币解决了这个问题。

随着比特币的出现，即时、去中心化、匿名的价值转移终于变成了现实。神秘的比特币缔造者，那个自称中本聪（Satoshi Nakamoto）的人，有效地解决了困扰密码研究几十年的**拜占庭将军问题**。这里引用定义拜占庭将军问题的论文（Lamport，1982）："（假设）拜占庭军队的一些将军率队在敌人的城市周围安营扎寨。他们相互之间只能依靠信使通信，而且必须在作战计划上达成一致。然而，他们中间可能会有一个或几个想要迷惑其他人的叛徒。那么我们要解决的问题是，找到一种算法来确保忠诚的将军能达成一致意见。"在比特币中达成去中心化的共识，意味着任何一方都无须信任参与信息分享的其他各方，也无须通过一个中央权威来分享信息，其中包括以价值交易形态存在的信息。

比特币和其他"加密货币"将有助于定义互联网的第 5 层协议，让机器像传递数据那样快速有效地传递价值。比特币是很有用的在线价值传递工具，但它最有价值的贡献是其革新性的底层技术：**区块链**（blockchain）。这一技术首次将去中心化共识变成了现实。

区块链是对发生在比特币网络中的所有交易进行大规模复制的数据库。它采用了一种称为工作量证明（proof-of-work）的共识机制，以此来防止在网络中出现双重消费（double-spending）。双重消费问题困扰了密码研究学者几十年，指的是坏人可以对第一次交易予以否认，从而达到将同一笔资金重复使用两次的目的。

工作量证明解决这一问题靠的是在网络中引入**挖矿机**（miner），用其硬件进行加密证明。挖矿机是验证交易的比特币网络节点，会通过自己的区块链

历史来检查交易。区块链历史包含所有曾发生在网络中的交易，是一条带有时间戳的记录。从理论上讲，区块链历史可以修改，但因为有工作量证明，还需要使用网络上的大部分计算力进行验证。因为目前比特币网络所拥有的计算力已经远超世界上所有超级计算机的计算力总和，所以攻破比特币网络极其困难。

从电力消耗和计算负载角度来看，工作量证明需要付出高昂的代价，但它是目前已知能够阻止女巫攻击（Sybil attack）的唯一机制。女巫攻击是指坏人在网络中宣称拥有多个身份，并获取他们不应有的资源来进行攻击。一次成功的女巫攻击极有可能导致比特币完全贬值，因为人们将不再相信它的稳定性。虽然工作量证明代价高昂，但到目前为止，它是唯一经过大规模验证的有效机制。

我们拥有了这样一个称作区块链的新工具，它是一个大规模复制交易数据库，能阻挡女巫攻击。区块链让我们第一次无须使用中心服务器就能达成去中心化共识。你可能想知道这有什么用，也确实应该知道。接下来，我要用很大的篇幅帮你考虑所有的可能性，以及实现它们的方法。不过眼下的重点是让你明白，有众多数据结构能帮你创建出可以盈利的去中心化应用，这只是其中的一种。

1.2　什么是去中心化应用

大多数人熟悉“应用”（application）这个术语是因为它与软件有关。应用软件是指定义了明确目标的软件。目前使用中的应用软件多达数百万，而绝大多数 Web 应用软件都采用集中式的服务器 – 客户端模型。另外有一些是分布式的，还有很少一部分新的应用是去中心化的。图 1-1 直观地展示出了这三种软件模型。

集中式系统是目前最流行的应用软件模型。集中式系统直接控制各个单元的操作，并且信息流源自一个中心。所有单元都要直接依靠中心点来发送和接收信息，以及接受命令。Facebook、Amazon、Google 和其他主流互联网服务用的都是这个模型。我们将这些巨型服务称为“服务栈”。这些服务栈很有用，因为它们为我们提供了有价值的服务。不过它们也有巨大的缺陷，我会在第 2 章展开讨论。

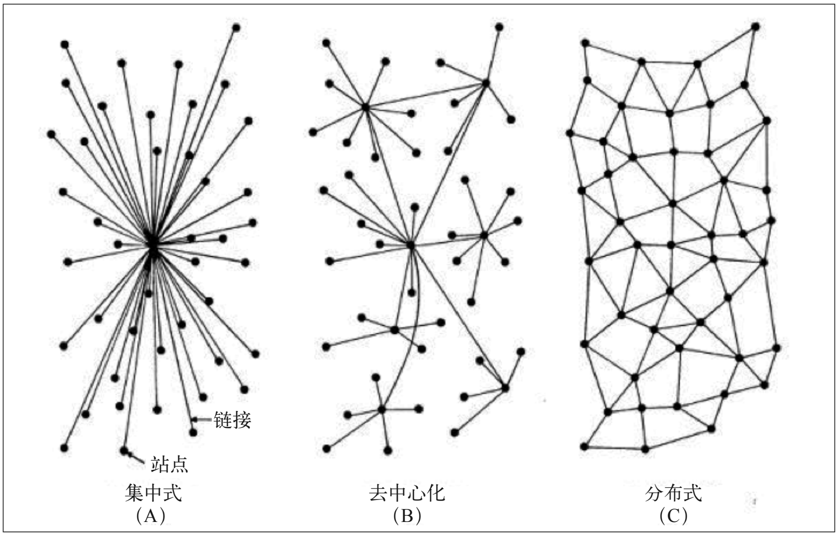


图 1-1：应用软件的三种类型

那么，去中心化和分布式有什么区别呢？

分布式意味着计算不是在一个节点上，而是分布到多个节点上完成的。去中心化的意思则是，任何一个节点都不会对其他节点的工作指手画脚。很多像 Google 这样的服务栈都在内部采用分布式架构，以加快计算速度，降低数据延迟。也就是说，集中式系统同时也可以是分布式的。

那么，去中心化系统可以是分布式的吗？

可以。比特币就是分布式的，因为它盖有时间戳的公共账目（区块链）就是驻留在多个计算机上的。同时它也是去中心化的，因为如果某个节点失效了，整个网络还可以照常运转。也就是说，任何使用区块链和其他端到端工具的应用都可以是分布式的去中心化系统。

那为什么本书不叫作《分布式的去中心化应用》呢？

集中式系统也可以是分布式的。能够达成去中心化共识的应用软件才是真正的革新成果。

那么，有去中心化共识是成为去中心化应用的唯一要求吗？

dapp 领域是一片刚刚开始开垦的沃土，有很多聪明人正在用新的模型进行各种尝试。对于究竟什么是 dapp，不同的开发人员有不同的看法。一些人认为只要没有能导致整个系统失效的中心点就够了，但也有人觉得还要加上其他要求。本书的重点是讨论能够盈利的 dapp，即能让开发人员和用户赚钱的 dapp。之所以关注盈利，是因为利润为成功、健壮、可持续发展的 dapp 奠定了基石。开发人员构建应用，用户保持忠诚，以及矿工维护区块链，都是靠激励措施维持的。接下来介绍所有能够盈利的 dapp 都应该具备的 4 个特性。

1.2.1 特性1：开源

去中心化的闭源应用要求用户相信该应用的去中心化程度确如核心开发人员所说，并且用户不会通过一个中心源来访问自己的数据。因此，闭源应用会让用户望而却步，不敢使用。尤其对于那些会收取、持有或转移用户资金的应用，闭源更让人排斥。尽管确实可以推出一款闭源的去中心化应用，但从一开始就会面临艰难的局面，而且用户会更加青睐开源的竞争对手。将 dapp 开源会改变它的商业行为结构，因此互联网才会变成共同点，而不是孤岛链（见图 1-2）。

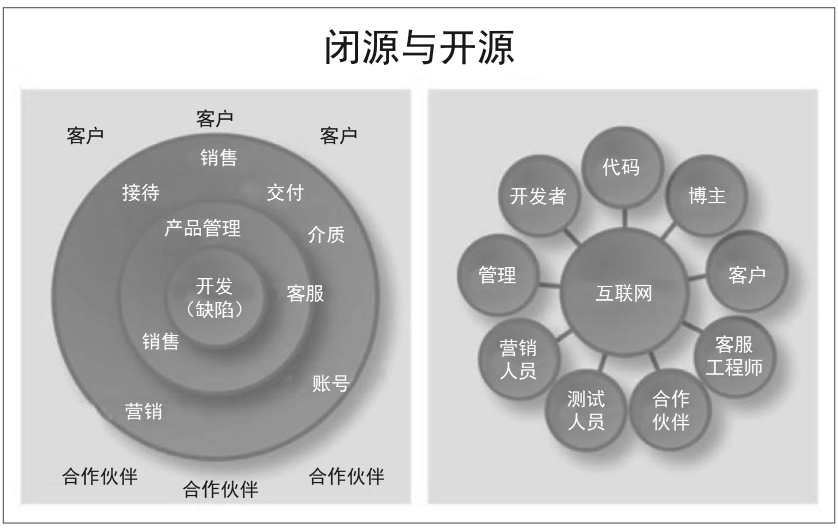


图 1-2：闭源商业计划与开源商业计划的对比

所有应用都能开源，但为什么不这样做呢？

如果研究一下传统的商业模型，就会发现它们全都要求所销售的产品或服务要超过竞争对手。如果把产品开源，竞争对手就能窃取你的工作成果，改头换面后当成他们自己的产品销售。

那么，是什么原因促使开发人员把希望从中盈利的应用开源呢？

从让开源 dapp 的创建者盈利这个角度来看，比特币树立了一个很好的榜样。中本聪保留了最初的一部分比特币，然后让其他人使用其余的部分。因为有数量上的限制，并且比特币网络的工作量证明机制为社会提供了巨大的价值，所以比特币的价值开始增长，从而为中本聪创造了财富。通过开源吸引开发人员为其做出贡献，该应用达成了网络自行完善所需的透明性，并赢得了用户的信任，最终使得比特币在现实世界中有了价值。开源 dapp 能让它赢得潜在用户的信任。任何人都可以从你的 dapp 创建分支，但他们挖不走你的开发团队。用户希望让最合适的人，通常也就是最初的那些开发者，来维护 dapp。

1.2.2　特性2：“内部货币”

在 dapp 圈，总会有人问怎么靠它赚钱。集中式应用的传统赚钱模式包括交易手续费、广告收入、推荐佣金、访问用户数据的权力以及订阅服务。如果把 dapp 开源，该怎么赚钱呢？你可能想通过程序自动产生交易手续费，并把这笔钱转到开发者的账户上去，但是可能会有人创建应用的分支，把你的佣金拿走，所以这样是不行的。嵌入广告、订阅服务以及其他任何集中式商业模型所采用的方法都是不可行的。

开源的 dapp 开发者要怎么赚钱呢？

答案是用稀缺令牌，即 App 币，来分配网络中的稀缺资源。用户如果想用这个网络，就需要 App 币。稀缺资源的所有者得到别人支付的 App 币。在比特币网络中，稀缺资源（计算力）的拥有者（矿工）直接从用户那里获取交易手续费，让他们使用自己提供的服务。因为网络的增长会引入更多用户，而 App 币的总额是固定的，所以 App 币的价值也会不断增长。我们可以把这个模型应用到所有 dapp 上。稀缺资源可以是存储空间、交易、图

片、视频、文本、广告等很多东西。

这是不是意味着用户使用任何 dapp 都需要付费呢？

既是也不是。尽管区块链是需要支付才能使用的，但在 dapp 内部可以构造出不同的激励方式。可以给用户 App 币作为注册奖励，甚至可以让用户通过出售自己的数据或本地存储空间来换取 App 币。除了使用 App 币，dapp 创建者还可以通过虚拟资产赚钱，就像去中心化的 MMORPG 中的不动产一样，或者通过特殊命名空间中的领域，甚至声誉。

1.2.3　特性3：去中心化共识

在比特币之前，达成交易有效的共识总是需要或多或少地依靠一个中心来完成。如果你想完成支付，那么这条交易必须经过一个清分中心（所有的交易都要受它监测）。比特币是点对点的（P2P），也就是说各节点可以直接通话。P2P 网络不是什么新发明，BitTorrent 之类的分布式散列表（DHT）是在区块链之前发明的。如果要存储和串流去中心化的数据，DHT 非常好用；但如果想以一种去中心化的方式让所有人对用户名、状态更新、高分值等应用层面上的数据达成共识，则需要靠区块链。区块链并不会替代 DHT，但确实是其必要的补充。区块链之所以如此独特，是因为它解决了 DHT 最主要的安全问题：节点之间不需要在数据的有效性上相互信任。区块链是一个去中心化的交易数据库，并且是第一个高度防篡改的去中心化数据库。安全性是区块链的主导性设计目标之一。它是历史上第一个在组织上去中心化而在逻辑上集中的交易日志。请看表 1-1。

表1-1：比特币在组织上去中心化，在逻辑上集中

	组织上集中的	组织上去中心化的
逻辑上集中的	PayPal	比特币
逻辑上去中心化的	Excel	电子邮件

区块链的革新点在于去中心化的共识。如果你的应用中有个功能是需要让所有人共同承认某个东西，就应该用区块链。以用户名系统为例，用户名 @user 在谁手里其实并不重要，真正重要的是让所有人都承认它就是那个人的。过去出现过很多去中心化协议，但都要求节点间彼此信任。区块链是不可变记录，每个节点都有一份副本，因此没有人能假装自己也是 @user。这可以用**智能合约**实现。

智能合约是区块链里的一段代码。当某个预先编写好的条件被触发时，智能合约就会执行相应的合约条款。你可能在想：“跟下面这个用 Stripe 的 API 所做的事相比，智能合约有什么不同之处呢？”

```
if (user.sendsMoney(customerID))
{
runContract();
}

func runContract()
{
 println('hello world');
}
```

一个很大的区别是智能合约就在区块链之中，不是放在服务器上。它不需要第三方信任，就这个例子而言，也不需要信任 Stripe 或者服务器的主人。因此，智能合约正规的表达应该是“用经济的加密方式保证安全的代码执行”。不过要记住，并非所有的 dapp 代码都是智能合约。尽管智能合约有自己特殊的用例，但为了便于展开讨论，可以说它们一般就像模型－视图－控制器 dapp 架构中的“模型”。在讲到 dapp 的架构时，我还会进行更深入的介绍。

1.2.4　特性4：没有中心失效点

dapp 是没有办法关掉的，因为根本没有服务器。dapp 的数据是去中心化地放在其所有节点上的。每个节点都是独立的；如果一个节点失效了，其他的还能在网络上运转。要在你构建的 dapp 上实现这一功能，可以选用一款去中心化数据库系统，比如星际文件系统、BitTorrent 和一些独立的 DHT。

1.3　去中心化应用的历史

在其早期，Web 上并没有这么多的应用和服务。当时明显不像现在这样全天下所有事情都能在 Web 上完成，但那时候分布式 DIY 的感觉确实更强烈。从一开始，Web 就是非常去中心化的。HTTP 协议连接着全世界拥有计算设备和互联网连接的每一个人。在 HTTP 协议的指导方针中，有一组可信的服务器将你输入的 Web 地址转换成服务器地址。HTTPS 更进一步，

加上了可信服务器和证书颁发机构这一层。人们可以架设一台个人服务器供他人连接，而每个人都拥有自己的数据。但很快，应用服务器开始起步，我们今天熟知的数据拥有权的集中式模型诞生了。为什么会这样呢？

简单来说是因为它很容易实现，从概念上和编程上来说都容易。这是最容易做到的，并且管用。个人或群组负担服务器的维护成本，并从使用服务器上软件的用户那里盈利。MySpace 和 Yahoo! 是第一批集中式应用的典型代表。最近的应用，比如 Uber 和 Airbnb，通过提供一个可信的集中式数据存储，把业务的“现实世界”部分去中心化了。他们是第一批允许各方全都参与到一项赚钱事业中的商家。这种去中心化的业务模型也为开发更多的去中心化应用做好了铺垫。

随着 HTTP 网络的不断增长，一个名叫 Bram Cohen 的开发人员引入了一种新的协议，称为 BitTorrent。Bram Cohen 创建 BitTorrent 协议的目的是解决通过 HTTP 协议下载大型媒体文件时间超长的问题，同时也是对之前 P2P 协议的一种改进，比如 Gnutella、Napster 和 Grokster。问题在于，下载大型文件所需的时间太长，并且随着 Web 的增长，能够获得的文件大小也在增长。与此同时，硬盘存储空间也在增长，连接到一起的人也越来越多。BitTorrent 把下载者同时变为上传者，从而解决了这个问题。

你想要的文件并不是放在一个数据源上，所以下载的时候是从多个数据源下载的。这个文件越受欢迎，就会有越多的用户下载它，从而也就有越多的用户上传。这就意味着，你能从更多的源头来拉取这个文件中的数据。数据源越多，下载速度越快。给种子用户的回报是更快的下载速度，而对吸血用户的惩罚就是速度上的限制。事实证明，这种“以牙还牙”的数据传输系统对于电影和电视剧之类的大型文件特别好用。

BitTorrent 不断增长，并且成为游戏和电影等所有大型文件的下载方式。事实证明，对于大型数据集来说，BitTorrent 的速度、弹性和回报机制都要优于 HTTP。

那 Web 为什么不用这种方式工作呢？

很有可能是因为 HTTP 的先发优势，它的基础设施，以及所有已经投入其中的时间和金钱。现在有些活跃的项目想要对 HTTP 网络进行升级，给它

加上类似于 BitTorrent 的技术。他们很有可能会成功，因为 BitTorrent 有巨大的价值主张。BitTorrent 刚一出现，开发者就开始用这一技术创建非盈利性的去中心化应用。下面来看几个最近出现的去中心化应用。

1.3.1 PopcornTime

PopcornTime 用 BitTorrent 协议实现用户间的实时视频流，有点像使用种子的 Netflix。它是美国电影协会（MPAA）最恐怖的噩梦。没有监管者能关掉它，现在所有人都能访问到免费的电影。作为去中心化版的 Netflix，PopcornTime 证明了自己是一个好用的 dapp。创建者声称每个国家都有人下载过它，甚至包括没有互联网的两个国家。因为 PopcornTime 没有使用“内部货币”，并且不需要去中心化共识，所以没有用区块链。它仅仅通过串流电影就提供了很大的价值。

1.3.2 OpenBazaar

OpenBazaar 的目标是成为去中心化版的 eBay。没有中间商能告诉销售商什么能卖，什么不能卖，也没有中间商能决定使用服务的费用。它构建在 BitTorrent 协议之上，但问题是销售商必须架设自己的商铺。他们需要有自己的服务器并保持运行，以便用户能看到他们的商品。在理想情况下，销售商只需要把店铺数据上传到网络上，还可能需要支付一小笔费用，然后就不用管了。这就要求有一个能够激励存储挖矿机的去中心化系统，我们会在第 4 章详细介绍。OpenBazaar 使用 BitTorrent 传输数据，用比特币作为销售商之间的交易“货币”。

1.3.3 Lighthouse

我们会在第 5 章深入探讨 Lighthouse，这里先进行简要介绍。Lighthouse 是一个内嵌了一系列智能合约的比特币钱包，这些智能合约就像 Kickstarter 一样帮忙向某些项目认筹。当项目目标达成时，它就能从项目支持者的 Lighthouse 钱包中取出资金。认筹者可以随时自行撤销认筹，不需项目创建者的参与。Lighthouse 是用现有的比特币基础设施构建自己的 dapp 的好例子。它只用一个带有一些比特币智能合约的 UI 就做成了一个电子钱包。此外，它运转良好，并且吸引了比特币现有的用户群。Lighthouse 有去中心化

共识，是开源的，没有中心失效点，但它没有发行自己的“货币”，而是用了比特币。它是个实用的 dapp，但不能让创建者盈利。

1.3.4 Gems

Gems 是一个社交消息应用，试图创建一个比 WhatsApp 更加公平的业务模型。Gems 发行了自己的“货币”，并且允许广告商直接向用户支付费用以访问其数据，而不是自己作为中间商盈利。这种货币称为“宝石”，是在比特币基础上创建的元币。用户吸引其他人加入网络能赚到宝石，开发人员在开发和维护这个软件时也能赚到宝石。随着 Gems 用户群的不断扩大，宝石的价值也变得越来越高。用户也像开发人员一样受到激励，通过扩大网络来赚钱。你可以把宝石看成 dapp 里的股份。Gems 还没有开源，所以用户无法检查它是否真的没有中心失效点。这是一个盈利性 App，但我觉得它难以跟满足另外三个条件的竞争对手进行竞争。

那么，有没有四个条件全都满足的独立 dapp 呢——没有中心失效点，发行自己的“内部货币”，有去中心化共识，并且是开源的？

有很多“加密货币”都是同时满足这四个条件的，但“加密货币”不是 dapp。我所说的是社交网络、拼车、搜索引擎：去中心化版的服务栈。答案是现在还没有，但是有这种可能性。目前存在这种技术，只要出现几个这样的 dapp，就会有大批开发人员走上这条路，让自己和他们的用户大赚一笔。接下来就来聊聊要用到的技术。

1.4 技术点

在讨论去中心化应用的发展历史时，我已经提到了很多技术点。比特币的区块链当然是最重要的，所以在介绍其他技术点之前，我们先深入探讨一下区块链。区块链帮助解决了拜占庭将军问题。该问题问的是：“如何在分布式节点间进行协作，从而能够在攻击者试图搞破坏的情况下达成某种共识？”工作量证明算法和区块链帮我们解决了这一问题。

在比特币被创造出来之后，去中心化共识成为了可能。工作量证明并不完美，它要消耗大量的计算力和电力。还有其他可以解决有意义问题的“加密货币”，比如 PrimeCoin，它的挖矿机用自己的计算资源寻找素数。在比

特币的世界中，我们将要耗费大量的能量来维护网络，而能量应该有更好的用途，只是帮助网络维护其自身的安全太浪费了。

但问题是，工作量证明是目前为止唯一能够防范女巫攻击的系统。共识研究并没有止步于工作量证明，探索仍在继续，但现在我们最好的选择就是工作量证明。至于说有希望成为工作量证明的竞争对手的，现在有个大家伙：**权益证明**（proof-of-stake）。权益证明也不完美，但可以作为工作量证明的补充。

权益证明是一种共识机制，依靠计算力防止对网络中的权益进行女巫攻击。通常来说，权益是指矿工拥有的“加密货币”数额。其核心思想是，你拥有的“加密货币”越多，就越会投入更多来确保网络的稳定性，也越不可能进行 51% 攻击来创建区块链的分支。代理式权益证明在权益证明的基础上做了创新，一组 101 代理可以对区块产生者进行投票。代理式权益证明和权益证明都还在研究当中，但如果能证明其中任何一个是长期安全的，就可以将其作为工作量证明的补充，甚至完全替代工作量证明。

定义术语

为什么要使用术语 dapp ？为什么是去中心化应用？为什么不是去中心化组织、去中心化自治组织或去中心化自治企业？

对于 dapp 这个理论上可行并已得到部分实现的生态系统，“加密货币”圈里已经有很多不同的术语了。要解释我选择术语 dapp 的原因，最好的办法是看一下目前所有相关的术语，以及它们的意思。

先从 dapp 本身开始吧。

- 去中心化应用（decentralized application，DA）
  去中心化应用是本书的标题。我本来也可以有其他选择，比如 DO、DAO 或 DAC。为什么最终选用 dapp 呢？因为所有这些术语里都有“去中心化”。对于所有涉及软件的去中心化实体，去中心化应用都是其超类。

- 去中心化组织（decentralized organization，DO）
  DO 是给所有员工授权的组织。这个术语并不适用于组织所用的工具，

更多的是描述其结构。组织的去中心化程度不同，并且完全去中心化也不一定是最好的方式。在传统型的组织中，有严格的层级式命令结构。

去中心化组织会倾听员工的声音，并且权力是平均分配给每个人的。每个人都可以审查公司的活动和里程碑，它们就存放在一个去中心化存储网络中，以便达到最佳的弹性。不需要把人类当作唯一的决策者：智能合约能够在某个日期担任诸如支付者的角色。DO 也不需要基于某座城市，其成员可以遍布全球。在某些系统中（比如比特币），共谋被视为 bug。但是在去中心化组织中，共谋是一项功能。得益于 Slack 和 GitHub 等工具的发展，最近有些创业公司采用了更加去中心化的结构。

- 自动代理（automated agent，AA）

  AA 不一定是指 SkyNet 或一些通用的人工智能。自动代理至少已经存在 10 年了，无须人工干预运行的或者说自主运行的软件都是 AA。计算机病毒就是个完美的例子。开发者制作并把它广泛散发出去之后，就由它来决定进行自我复制或执行已编码的任何其他维护算法。守护程序是另外一个例子。守护程序会作为后台进程在操作系统中运行，比如电子邮件程序。自动代理有利有弊：它们确实不需要任何维护，但未经检查的的代理也可能变成人类难以控制的危险源。在第 6 章还会对其进行详细讨论。

- 去中心化自治组织（decentralized autonomous organization，DAO）

  在换成 dapp 之前，我最初其实是想用它做书名的。DAO 跟 DO 一样，只是由 AI 代替人类来做决定。协议就在去中心化的栈中，并且不会听从任何法律约束。人类不是主宰者，而是被边缘化了。AI 来做决定，DAO 自己来维护自己。但并不是由 AI 做决定的就是 DAO，它还要有自己的内部资本。

  简言之，这些都是 dapp 的子类，并且 DAO 是由 AI 控制决策的 dapp，人类靠边站。共谋不像在去中心化组织中那样被当作功能，而是 bug。比特币就是 DAO 的一个例子。

- 去中心化自治企业（decentralized autonomous corporation，DAC）

  这个有争议。一些人觉得这根本不应该是个词组，因为企业这个词是从受法律合同和层级集权控制的遗留系统中来的，而这样的系统正是我们要努力演变的起点。另一些人认为 DAC 是 DAO 的子类，将股息付给其成员。

我支持前一种观点，因为我不喜欢企业这个词，并且如果 DAO 想要实现给人类和机器成员的派息，它可以是 DAO，而不是 DAC。

我们讨论了 dapp、DO、DAO、AA 和 DAC，每个还都举了例子。接下来再看一下图 1-3，以便了解得更清楚。

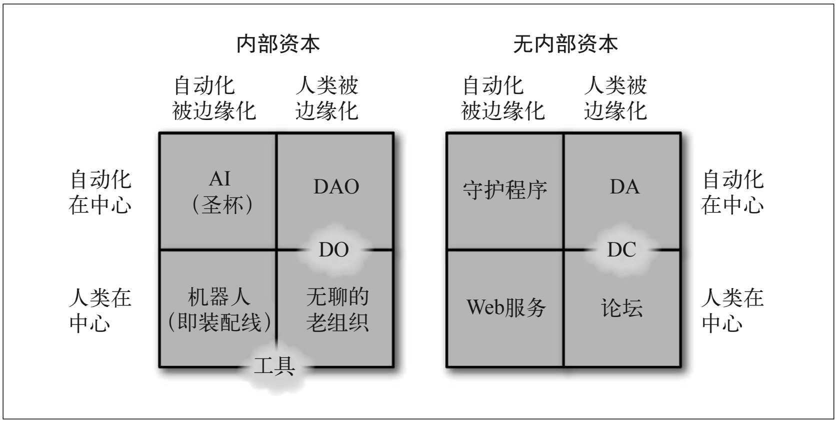


图 1-3：组织的类型（来自 Vitalik Buterin）

我很喜欢这张图，因为它把之前讨论的内容都放进了上下文中。我们还没到能够制作 AI（图中的圣杯部分）的阶段，而是在旁边那个可以开始制作 DAO 的进化阶段。

为简便起见，我们将在本书中使用术语 dapp，因为 dapp 是所有去中心化软件的超类。我将要讨论用于定义你自己的 dapp 的各种工具及方法论，然后由你来决定哪种 dapp 最适合你。

我的定义源自我对“加密货币”社区的研究。我不想给概念添加其他标签，也不想创建新的范式。实际上，我想要尽可能地简化这一领域，以便你能充分了解那些用来创作可盈利去中心化应用的工具。去中心化应用领域几乎快被各种想法搞得精疲力尽了，在看过它的优缺点后，是时候重新迭代一下了。下一波软件潮流就是 dapp，希望这本书能帮你做好加入其中的准备。

1.5　开始着手吧

希望我已经把什么是去中心化应用讲清楚了。虽然还有很多需要解释的，但你应该已经对这一领域有了大致的了解，并且掌握了跟 dapp 相关的术语和缩写。我写作这本书的目的是：首先解释什么是 dapp，为什么要构建它们，以及蓬勃发展的 dapp 生态系统看起来是什么样的；然后介绍如何使用现有工具实现你自己的 dapp；最后深入介绍 dapp 领域中的几个主要参与者。

第 2 章

蓬勃发展的dapp生态系统

区块链这个领域很容易把人搞糊涂。似乎有数不清的创业公司、新币种、意识形态和流行语层出不穷，很难把它们全都弄清楚。根据 Melanie Swan 的《区块链：新经济蓝图及导读》一书和其他人的观点，可以把这个领域细分成三类：区块链 1.0 是“货币”；区块链 2.0 加上了合约（股票、债券、金融资产）；区块链 3.0 超出了纯粹的金融领域，涵盖了治理和健康之类的应用（dapp）。我们将在本章探讨这三类区块链靠什么来推进。作为一名 dapp 开发者，你只需要知道一点：那些能让你的 dapp 变得安全、健壮、可盈利的工具。本章将会描述蓬勃发展的 dapp 生态系统，也就是非常容易制作 dapp 的生态系统。我还会讨论制作 dapp 所需的技术，以及目前最可行的方式是什么。

一直以来，Web 应用中有四个概念是处在集中控制领域中的：身份标识、财富、数据和计算。其中每一个都要对服务提供商有充分的信任，然而这种信任却有可能遭到背叛。最近，在分布式系统中出现了一些新技术，让用户可以控制这些事情。接下来就让我们研究一下这些创新。

2.1　去中心化数据

对我来说，这是最重要的概念。现在，我们放心大胆地把自己的数据交给“服务栈”，心甘情愿地用自己的数据换取他们提供的免费服务，甚至会因

为委托他们存储数据而支付费用。但是实际上，只要用户免费把数据给他们，他们就能靠这些数据把存储数据的钱赚出来。我们相信提供商不会滥用我们的数据，也不会把数据卖给那些我们不愿意向其暴露这些数据的人。然而现实是，我们知道，只要把数据委托给一个集中式实体，这份信任就可能会被辜负。Amazon 的 Web 服务、Google 云端硬盘、Dropbox 以及其他任何一家“云”服务提供商，尽管有着分布式的计算后台，但所有权都是集中的。

另外，随着机器人和自动化技术的急速扩张，全球经济正从以劳动力为基础快速迈向以信息化为基础，数据将变成价值的主要形态。尽管人类的劳动力跟机器人没法比，但在数据上却可以一较高下——这是利用对世界的独特看法而分析出来的数据，是通过五种感官处理过的输出。我们不仅要拥有数据，还要在日新月异的世界中掌握其所有权。

那该怎么解决这个问题呢？怎么把数据以一种去中心化的方式存储，让你独自拥有自己的数据？这个问题至少已经得到了 10 年的深入研究，并且已经由几方提出了一个解决方案。理想的方案应该是提供一种去中心化的数据存储方式，它要足够健壮，并且尽可能不需要依靠信任保护数据。

2.1.1 方案1：把数据直接存放在比特币的区块链中

这种方法比较幼稚。它确实把数据去中心化了，因为每个人都有一份存储数据的区块链副本，但谁也不能修改数据。数据当然会用 SHA-256 加密，每个有钱包的人都会存储一份数据副本，但只有掌握了私钥的你能够访问。不过比特币的区块链不适合用来处理大量数据！区块链的设计目标是存储简单的交易日志，这个任务它完成得很好。但即便是只存储交易日志，区块链的规模在过去这几年就已经超过了 38 GB，下载一次可能要花上几天时间。核心开发人员一直密切关注着扩展性和区块链膨胀问题。在你把数据上传到区块链之后，比特币矿工只能免费存储你的数据，他们得到的报酬根本不足以负担支出的成本，所以也就没有了继续维护比特币网络的动机。

专门用一个放宽大小限制的区块链来单独存放额外的数据怎么样？即便用另一种“加密货币”作为报酬支付给为你存储数据的矿工，这种办法还是

不行。因为区块链会疯狂增长，任何想要使用这种“货币”的人都要下载超级大的钱包。仅仅几个用户存储一些图片就会使其臃肿不堪，更何况我们即将步入 PB 级数据都很平常的时代。要获得健壮的去中心化数据存储，不管从短期还是长期来看，都不能把数据存在区块链中。

2.1.2 方案2：把数据存放在分布式散列表中

分布式散列表（DHT）在过去 10 年里得到了广泛采用。它们不仅分发数据副本，还包含查找数据的索引函数，可以确保弹性。像 KaZaA、Napster 和 Gnutella 这些早期的 P2P 文件共享程序用的都是自己的 DHT，去中心化程度各不相同。一些用中心追踪器来监测所有数据的移动，一些（比如 Napster）有所有数据都要通过的中心源，它们都有单一失效点（出于法律原因）。

真正把 DHT 发扬光大的是 BitTorrent，它现在仍然有 3 亿多用户。尽管有去中心化的数据存储（BitTorrent 主干 DHT），BitTorrent 仍然要靠中心追踪器（比如海盗湾）来监测网络。海盗湾这样的网站会由于法律原因被定期关闭，所以即便 BitTorrent 有数据弹性，还是会有一些失效点。如果我们用 BitTorrent 的 DHT 存放 dapp 的数据，是不是很好？ BitTorrent 不仅提供了去中心化数据存储，还提供了一种数据分发协议，通过在种子用户和吸血用户之间设置对抗性策略使带宽的利用率达到最大化。

BitTorrent 的数据传输协议甚至比 Web 的还要快，因此它成了通过 Web 传输大型数据集（如高清电影）的主流方法。但用 BitTorrent 存储数据也有问题，各个节点没有长期为你保存数据的动力。在 BitTorrent 网络中，需求越旺盛的文件优先级越高，所以只有在获得人们的需要时，你的数据才会一直被复制并留在网络中。然而在声誉良好的中心服务器上，比如 Amazon 的 Web 服务之中，即便只有你一个人用，数据也会一直放在那里。为了保护自己的声誉，他们只能依照合约保存好数据，不会因为没有其他人使用而不再保存。

首先，我们要的不仅仅是 DHT 的去中心化存储能力和 BitTorrent 的文件传输速度——我们还想要持久保存数据。因此，必须以某种方式激励节点存储数据。另外，我们需要保证指向数据的链接不会挂掉。互联网最初的提

案中就有一条是链接的永久性。这个想法源自上都[1]项目（Project Xanadu），在其所描绘的 Web 中，每个链接都有两个方向：一端指向目标，一端指向它的源头。也就是说，内容的创建者总能获得创建数据的认定，因为所有链接都会链回到他们。然而这样的 Web 始终没能出现，所以我们现在用的是基于 HTTP 的 Web，伴随我们长大、让我们因熟悉而喜欢的也是单向链接。

有没有哪个系统实现了这些功能呢？有，它叫星际文件系统（IPFS，http://ipfs.io）。这个开源项目目前仍处于 Alpha 阶段。我非常喜欢 IPFS，并且是其协议的早期贡献者之一。它的创建者 Juan Benet 曾经花了 5 年时间思考数据存储问题，并最终付诸行动，发布了 IPFS 科学论文（http://bit.ly/ipfs-whitepaper）来阐述所有的想法。我用了几个月的时间来了解这个系统和他的思想框架，思考为什么 IPFS 比其他方案好。目前，我觉得它最有可能成为最有价值的数据存储方案。

IPFS 致力于帮我们发展出一个永久的、去中心化的 Web，一个永远不会有死链的 Web，一个数据不再由单一实体控制的 Web。下载好 IPFS 客户端后，用户能用它向网络上添加任何数据，然后会得到一个散列值。之后，用户就可以用这个散列值访问对应的数据了。跟基于 IP 寻址的 Web 不同，IPFS 是一个内容寻址系统。在 IP 寻址系统中，如果命名服务器失效，它的所有数据也会失效。内容寻址的寻址数据则高效得多，因为用它访问数据不需要依赖单个服务器的运行。从一个内容地址请求数据时，得到数据的速度要比从 IP 地址请求数据更快，因为它会根据内容地址路由到离你最近的数据副本。

从后端来看它是什么样的？

IPFS 用 DHT 保存数据。它基于流行的 Kademlia DHT，还借鉴了 Chord 和 BitTorrent 的 DHT。用户上传到 IPFS 的数据会被复制到几个节点上，所以即便某个节点失效了，数据依然可以访问得到。除此之外，就像 BitTorrent 一样，需要某份数据的节点越多，数据的弹性就越大，因为所有下载那份数据的节点都会分享它们持有的副本。Chord 的顶级功能是它的 DHT 圈，

注 1：位于今内蒙古自治区锡林郭勒盟正蓝旗境内，多伦县西北闪电河畔。因为马可·波罗的介绍，它在西方渐渐被引申为世外桃源的意思。——译者注

它会创建和弦（chord），并将相互靠近的和弦扩展为更大的和弦，从而使 DHT 查询在全局节点内最大化。这样，网络全局看起来就像一系列不断增大的和弦（见图 2-1），而查询将会得益于其效率，在必要的地方进行和弦之间的跳跃。

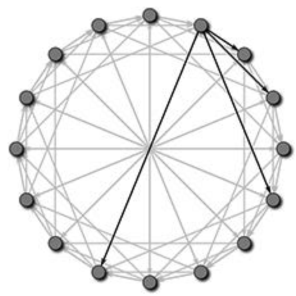


图 2-1：和弦

现在，像 Amazon 和 Google 这样的集中式服务提供商都有遍布全球的数据中心，用户可以自行选择用哪个数据中心来接收和路由自己的数据，但服务一般都会自动帮你选好。即便跟遍布全球的数据中心比，类似 Chord DHT 这样的系统还是有优势的：它们可以提供一种特有的办法，用多个节点来提升数据的传输效率。

IPFS 用 merkleDAG 作为 DHT 的结构，让用户在需要时找到数据。merkleDAG 是一种简单灵活的数据结构，你可以把它理解成一系列相互连接的节点。说得具体一点，就是一个有向无环图（DAG）。merkleDAG 看起来可能像一个链表或一棵树。往 DHT 上添加数据时，系统会生成一个 SHA-256 多重公钥 – 私钥对，然后把它们两个都交给用户。开发人员可以通过编程将散列值链在一起，形成他们自己的微 merkleDAG。一定要注意，IPFS 中的所有数据会形成同一个包含所有节点的泛 merkleDAG。IPFS 上的所有数据都是公开的，所以用户要自己负责数据的加密。私钥除了可以用来访问数据，还能证明所有权。

IPFS（如图 2-2 所示）受到了 BitTorrent 的启发——数据传输速度，为了寻找用于分享数据的同伴而采用的对抗机制。IPFS 团队相信 Web 也应该采用

这种工作方式。举个例子，如果一所大学里整个班的学生都跟 Facebook 的中心服务器上请求同一个视频，就会占用很多不必要的带宽，还会产生很多不必要的数据冗余。如果图片就在附近，他们没必要向那么远的服务器发起请求。在内容寻址系统中，如果知道所需数据的内容地址，就可以从最近的地方获取。节点间的数据分享不需要中心来协调。它跟 BitTorrent 一样，采用了 HTTP Web 所用的服务器－客户端架构，并且也做成了分布式的。

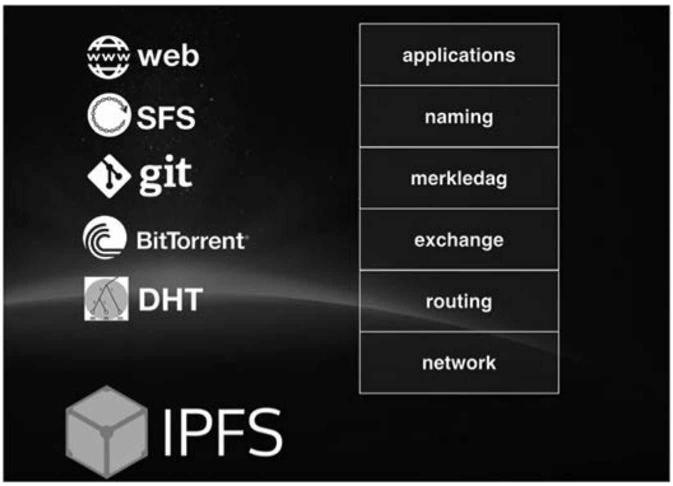


图 2-2：IPFS

IPFS是如何在BitTorrent上进行改进的

IPFS 有个姊妹协议，叫作“文件币”（http://filecoin.io/filecoin.pdf）。“文件币”用于支付给矿工（存储数据的节点），它采用了一种称为 BitSwap 的新型“价值换数据”机制。“加密货币”在这里发挥了作用：它的价值转移很快，并且允许根据每个相关字节的存储进行微支付。“文件币”目前仍在开发当中，但 IPFS 已经可以使用了。IPFS 命令现在是免费的，矿工存储数据完全出于他们对网络的热爱。最终，所有的上传和下载都会需要“文件币”。“文件币”很有可能成为直接建立在比特币区块链上的“资产”，因此用户可以用比特币“购买”存储空间。

除了上面这些，IPFS 还借鉴了 Git 管理所有数据版本的版本控制模型。Git

用 DAG 为数据版本建模，IPFS 用其作为整个系统的结构。用户能看到数据的版本历史（或者任何已经获得解密访问权限的数据）。

因此，IPFS 是 Git、DHT、SFS、BitTorrent 和比特币的集大成者，用这些系统最优秀的思想创建了一个去中心化数据存储网络。IPFS 希望有朝一日能将 Web 的 HTTP:// 协议换成 IPFS://，但它们也能协同工作。在谈到具体实现时，我会详细介绍几种方法。

IPFS 是经过深思熟虑的去中心化存储解决方案。尽管这一领域还有其他表现不错的竞争对手，但它是最健壮的，优于所有的“加密货币”项目。下面来看一下其他方案。

- *以太坊群*

  以太坊（https://ethereum.org）致力于构建一个通用（图灵完备）的区块链计算语言，包括去中心化存储。在写作本书时，他们的工作重心是确保 DAO（他们指的是“民主自治组织”）的安全，存储则放在了次要位置。

- StorJ

  StorJ（https://storj.io）最近被炒作得很厉害。它已经预先开采了很多 StorJ 币，并做了一些漂亮的设计。它的设计很整洁，在奥斯汀黑客马拉松上赢得了胜利，而且看起来开发小组的人也知道自己在讲什么。然而不管怎样，在黑客马拉松结束了一年多之后，它还是个雾件。

- Maidsafe

  Maidsafe（https://github.com/maidsafe/Whitepapers/blob/master/Project-Safe.md）跟以太坊一样，想做的事情有很多。他们没有使用工作证明，目标是为计算、存储和“货币”创建一个去中心化平台。他们已经在这个平台上工作了 6 年，但看起来还没得到足够多的关注。

2.2　去中心化财富

比特币是第一个成功的去中心化财富存储。在比特币之前，通过 Web 传输价值时需要一个可信的第三方提供商（银行）。比特币实现了去中心化价值传输，满足了 dapp 内部对去中心化支付的需求。

那些“山寨币”怎么样？莱特币、狗狗币、点点币、暗黑币和肯伊币怎么样？山寨币一般是从比特币的源码中分化出来的，添加了一些由于种种原因被比特币的核心开发人员拒绝采纳的功能。比如说，莱特币（https://litecoin.com）创建者想要提高支付速度，于是复制了比特币的代码，添加了一些加速代码。莱特币就由此诞生了。

莱特币的市值相当大，至少已经在“加密货币”前五名的位置上待了一年。莱特币是极少数出于好意（更快的支付速度）的币种之一。大多数山寨币就不一样了：如果不是让搞笑成为其支撑资金的模因（比如狗狗币），就是玩了“吸引散户接盘”的套路。山寨币的思想是，人们可以创建一种新币，给它贴上标签，然后通过媒体曝光来哄抬它的价格（肯伊币）。他们鼓吹说这个“加密货币”将来会非常值钱，早期买入的投资者都会大赚一笔。

这个新币的价格一旦高到一定程度，创建人就会把它全部卖掉，换成更稳定、更长久的货币，比如法定货币。这是山寨币圈子里的常见套路，明显会对“加密货币”的生态系统造成极其恶劣的影响。首先，这些山寨币会玷污“加密货币”的名声，让潜在的投资者变得越来越谨慎。其次，它们在毫无必要地与比特币区块链争抢市场份额，却不能带来丝毫真正的价值。这反过来又会损害比特币的价值，而且所有将比特币作为最常用“加密货币”的系统也都会深受其害。

当然，比特币使用的是工作量证明方案。也就是说，网络中的每台挖矿机都必须生成一个反映其计算力的计算证明，并负责处理事务。作为回报，挖矿机会得到比特币，作为它们维护网络的报酬。“加密货币”界的一些人觉得工作量证明消耗的能源太多了，并且只是防范女巫攻击的短期方案，所以在共识机制上展开了大量的研究。工作量证明用了大量计算力，并且挖矿机维护网络所消耗的电力成本高达 1500 多万美元。如果能找到更好的网络维护方法，就不用这么浪费了。有两种工作量证明的替代方案比较受欢迎，分别是权益证明和代理式权益证明。

尽管在比特币之后进行了大量的共识机制研究，比如权益证明，但目前还没有什么能像工作量证明那样抵抗得住女巫攻击。虽然计算昂贵，但目前来看没有比它更好的选择了。比特币网络上的投资已经超过 30 亿美元，有难以计数的创业公司、投资人、媒体和零售商接受了比特币。它有先发优势，并

且已经通过 5 年多的奋斗在一些国家或机构中赢得了认可。我们不需要重新开始。即便发现了优于工作量证明的共识机制，也应该让比特币的核心开发人员实现它，而不是交给某个山寨币。这样整个社区才能更快前进。

有些人可能会说这是“比特币最高主义”。反对者认为比特币最高主义者一直在鼓吹比特币的先发优势，并且认为他们为了保护自己在比特币网络上的投资，坚决反对任何竞争者出现。比特币最高主义的负面影响就是，不管多么有价值，任何不在比特币协议范围内的想法都会被迅速淘汰，得不到社区应有的认可，相关工作也会停滞不前。

这个问题有个两全其美的解决办法：**侧链提案**。该提案基于 Adam Back 与人合作的一篇论文[2]。Adam Back 是工作量证明的发明者，中本聪曾在其关于比特币的论文[3]中提到过他。这个提案源于一个想法：要测试共识机制以及任何关于“加密货币”的新想法，开发人员必须创建比特币区块链的分支，做一个全新的山寨币来验证其假设。

这对比特币没什么好处，并且对开发人员而言，要发起一个新的区块链是很困难的。巴克团队提出的解决方案是通过代码让比特币在主链（比特币区块链）和侧链之间自由切换。也就是说可以创建一个全新的区块链，并很容易地将它作为比特币区块链的侧链。你无须发起自己的挖矿网络，就能得到比特币工作量证明的安全保障。那些已经有“加密货币”投资经验（拥有比特币）的人会成为你的潜在客户群，因为他们可以直接使用自己手里的比特币。最后，在两个链之间发送比特币不需要任何转换。双向侧链现在正在开发，很快就会发布。

比特币区块链是相对安全的区块链，因为世界上所有超级计算机的计算力加在一起也不如它多，所以它对女巫攻击的防御能力也是最强的。从头开始发起一个工作量证明区块链非常困难，因为早期的计算资源太少了，攻击者很容易凑够 51% 的计算力从而接管整个网络。除此之外，开发人员应该专心构建用户需要的去中心化应用程序——这已经是个很有挑战性的任务了，不应该再分散精力去从头开始发起一个区块链。如果你想试验共识机制或者实现一些新的“加密货币”技术，侧链可以帮你。

注 2：https://blockstream.com/wp-content/uploads/2014/10/sidechains.pdf

注 3：https://bitcoin.org/bitcoin.pdf

如果你并不想实现新的“加密货币”技术，而是只想为自己的去中心化应用程序发行一个“内部货币”，该怎么办呢？如果要让这种“货币”可以跟随网络一起增值，允许用户访问稀缺资源，并且激励他们一起扩大网络，该怎么办呢？这样的话，你无须创建新的（侧）链，只要直接在比特币上创建一种资产就可以了。尽管在这种情况下也有好几种选择，但我会选择彩币。

- Counterparty

  Counterparty（https://counterparty.io）是比特币 2.0 的协议，让用户创建和管理资产、制定拍卖、出价，甚至在比特币上创建图灵完备的合约。听起来是不是很棒？但问题是 Counterparty 把这些有趣的特性全都放到了协议中。它不是模块化的，没有分层按顺序排好。在比特币区块链上发行资产并允许用户用比特币轻松传输是个很好的想法，但它们却跟股息功能合并到了一起。分配股息是个挺好的小特性，但却是 Counterparty 内部的操作，不是用本地比特币追踪资产。所有的事情都被迫揉进了一个过度雕饰的协议中。

  举个例子，投注就是在资产上添加了一个具有实验性、挑战性的特性。更好的做法是，构建简单的几层，每层都完成一件事情。模块化是优秀软件的标志。虽然 Counterparty 的野心很大，却根本没有做到模块化。假设有一个协议类库市场，所有协议类库都会相互竞争，无疑是最优秀的最终胜出。可以想象一下，如果所有类库都令人绝望地交织在一个包里，你要么把它们都装上，要么一个也不装。这简直就是一场噩梦，而这恰恰就是 Counterparty 要迫使你做的事情。

  Counterparty 中有一个 XCP“货币”。这是一个令人迷惑不解的元素，没有人想要它。如果你想用 Counterparty 的 API 构建一个应用币，就要应对 XCP 以及所有转换操作。如果要创建资产，不管它是什么类型的，都需要销毁 0.5 个 XCP（按当前价格计算要超过 1 美元）。XCP“货币”的供应是固定的，并且因为只要有人发行新的资产就要销毁“货币”，所以它的“货币”总量一直在减少。

  在使用 Counterparty 的某些特性时需要用到 XCP 让开发人员觉得很烦。这意味着你得一直盯着牌价（XCP/BTC）。虽然有平台在追踪这一价格，并且能跟所有市场一样提供实时报价及流动性需求，但说真的，这有什

么意义？你只不过要给你的应用创建一个“内部货币”，为什么要应付这么多麻烦事？这基本上就是一个没必要去跨越的巨大障碍。因此，这是个糟糕的主意。

Counterparty 一直在更新其客户端，那些依赖它的 API 的人，比如 Gems 应用（http://getgems.org），因此得到了一个混合的结果。因为没有模块化，所以如果出现 bug，一切都会马上崩溃。总而言之，Counterparty 过于集中式管理了。好在我们还有更好的选择（彩币），它们提供了必要的模块化和去中心化，也无须使用额外的“货币”。

- Hyperledger

  Hyperledger（https://www.hyperledger.org）相信自己是“代币无关”的，即允许发行者不用基于底层货币来发行新币种：既不需要比特币，也不需要法定货币，更不需要其他任何山寨币。它在理论上是健全的，但在实践中却不是，因为它靠的是一个籍籍无名的共识机制。在这一领域有很多研究，然而还没有哪个能够证明自己足以跟工作量证明相抗衡。

有种办法可以轻易切断任何一个区块链 2.0 项目发出的噪声，那就是深入探究它们的共识机制。如果没有使用工作量证明，或者不是基于比特币的，就看看它们的市场份额有多大，再看看出现过多少次安全漏洞。我每次这么做都会发现安全漏洞。表 2-1 汇总了这一领域的各种信仰，其制作者是 Meher Roy。

表2-1：“加密货币”的政治信仰

信仰/投注	平台机会	增量风险	优点
一级	不适用	不适用	不适用
代币不可知论	超总账、Eris、Codius、瑞波币 / 恒星币	缺乏身份标识和私钥管理的方案 因最终用户控制交易而导致的监管不确定性 平台特有的缺陷，如弱共识算法	适用于所有资产，包括平价货币、股票和“加密货币” 可以复制由“加密货币”社区率先开发的所有应用程序 与现有法规相对兼容
“加密货币”最高主义	比特币、以太坊、嫩薄荷、卵石币、瑞波币 / 恒星币（部分）等	对于新的价值形式，改变社会惯性需要大量的网络效应 拥有稳健货币政策和共识方法，交易速度快，并且具有可伸缩性的系统出现较晚	对传统银行体系不满意的细分市场是一个现成的市场 当前有重大的公共利益

（续）

信仰/投注	平台机会	增量风险	优点
比特币最高主义	侧链	那些比比特币强，能够改善网络维护成本、交易速度和可扩展性的新技术	比特币显著的先发优势
超比特币	不适用	被证明是幻想	无

代币不可知论是一套强有力的观点，但我相信比特币可以跟现有的金融系统合作。我们已经度过了比特币的蜜月期，同时也意识到，不管银行系统多么老旧，它在世界上确实有一席之地。

图灵完备的智能合约怎么样？这是在 dapp 中创建去中心化支付系统时不可缺少的金融工具中的第二部分。以太坊团队在这方面取得了最大的成功，但他们还有更大的野心。以太坊要创建一个图灵完备的区块链、一个去中心化存储网络、一个去中心化通信协议、一个运行以太坊 dapp 的新型浏览器，以及一种编写以太坊 dapp 的新型脚本语言。

我们稍微退一步来看。一个团队不能、也不应该想要单独完成所有这些想法，毕竟每个想法都是公司量级的。以太坊筹集了很多资金，也获得了很多关注，但是就算有创始人 Vitalik Buterin 的聪明才智，也不能指望其创造出下一代比特币。就像比特币协议的首席开发者 Gavin Andresen 所说，他们将来要么被安全问题搞得焦头烂额、疲于应付，要么会大规模缩减其区块链。

图灵完备的脚本语言是个好主意，你可以用它做任何想做的事。比特币的脚本语言特意做了限制，以防止无限循环这种恶意脚本出现（无论是出于恶意还是无能）。Gavin Andresen 说以太坊的大部分目标都可以用比特币实现，而且核心开发人员已经开始动手实现其中一些特性了。

考虑到 dapp 和本书的范围，我们接下来只讨论资产创建和智能合约；投注、衍生品和协议这些要求另一种货币的东西不在我们的关注范围内。从务实的角度来讲，要想让人们使用你的 dapp，最简单的办法就是确保这个 dapp 能用他们已有的“货币”，而目前份额最大的就是比特币。所以，你应该在比特币或者侧链上发行一种彩币，因为侧链基本上就是带有额外特性的比特币，比如更快的交易速度。

2.3 去中心化身份标识

身份标识的概念已经被人们争论了几个世纪。到了互联网时代，这个词有了全新的意义。什么是身份标识？谁拥有身份标识？在互联网上应该如何看待身份标识？

由于密码学最近的发展，很多解决方案都已经“假定有了一个公钥基础设施”。基本上，如果人们愿意安全地存储一个私钥，身份标识就变成去中心化的了。只有那些有密钥的人才能访问它。BitAuth（https://github.com/bitpay/bitauth）是一个现成的好例子。BitAuth 用比特币现有的技术创建一个使用 secp256k1 的公 – 私钥对。不需密码就可以进行跨 Web 服务的认证。它会给你一个系统识别码（SIN），也就是公钥的散列值。它用签名防止中间人（MITM）攻击，用随机数防止重放攻击。你的私钥永远不会暴露给服务器，你可以安全可靠地保管好它。身份标识是去中心化的，所以不需要把它交给一个可信的第三方，而是可以自己保管。

另外还有一些合并互联网身份标识的尝试，都取得了不同程度的成功。其中最值得注意的就是 OpenID（https://openid.net）协议。OpenID 是一个利用 HTTP、SSL 和 URI 等已有 Web 协议的去中心化身份标识协议。它的核心思想是，身份标识已经像碎片一样分散在 Web 上了，通过使用 OpenID 协议，用户可以将现有的 URI 传输到一个账号中，而这个账号在所有 OpenID 支持的网站上都可以用。

OpenID 将服务提供商需要存储身份标识的需求进行了抽象，让你可以只用可信的存储源，然后将身份标识带到多个提供商那里。在身份标识合并的尝试中，OpenID 是目前来看最成功的一个：Google、Yahoo! 和 Twitter 都已经是 OpenID 的提供商了。这种方式很好：我们无须重复注册就可以将自己的身份标识带到各个网站上，也就不需要一次次地重复输入身份信息了。这样不仅更方便，我们也不会因为要把身份标识数据存储在新的服务上而提心吊胆。不过 OpenID 仍然有潜在的安全隐患，因为你还是要把自己的数据托付给这些服务提供商中的一个。

这个问题又被称为 Zooko 三角（见图 2-3）。名称币（http://www.namecoin.info）旨在帮我们解决这个问题。

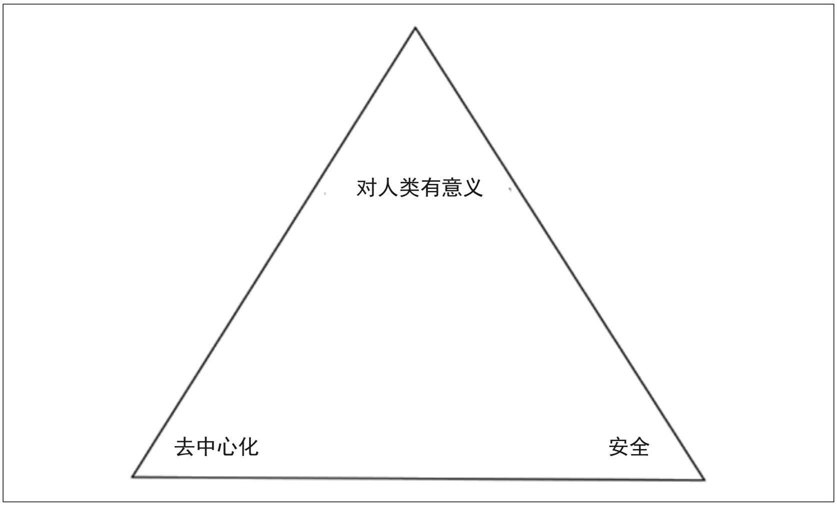


图 2-3：Zooko 三角

Zooko 三角是一种猜想。它指出，在一个按某种协议给出名称的系统中，只能实现其希望达成的三个特性（对人类有意义、去中心化、安全）中的两个。OpenID 实现了安全性和对人类有意义。名称币做了补充，添加了去中心化。名称币基本上是一个第三方身份标识提供商，由区块链作为你和请求身份标识的服务之间的中介。名称币的区块链是比特币区块链的早期分支之一，它的价值已经经过了时间的检验。

大多数山寨币都已经烟消云散了，名称币能留下来是因为只有它补足了 Zooko 三角。用户可以向名称币的区块链发送一条交易来注册自己的名称——他的名称会嵌在交易中，放在命名空间 /id 下。当用户发送交易时，如果是唯一的（之前没有人发过），名称币就把它存下来，否则不存。也就是说，只要人们能想到名称，命名空间就可以存。尽管这样用户可以创建和选择自己（人类易读）的标识，但由于人类易读的短语是有限的，这也是个问题。标识的分段确实有用，因为在新服务中，用户可以从新的命名空间中选择身份标识。

换句话说，这是个折中方案：让一个通用的身份标识提供商补足 Zooko 三角是一项重要创新，但命名空间是有限的。不过域名注册也是这样的。现在是 ICANN 控制着域名注册，这是一个得到美国商务部支持的集中式组

织。（它已经在 2016 年 10 月 1 日正式取得独立地位。）名称币由于提供 .bit 域名的注册而大受欢迎，变成了相对于 ICANN 的去中心化选择。Chrome 或 Firefox 之类的常规浏览器无法访问这些 .bit 域名。要进行访问，目前只能通过 .bit web 代理，或者下载插件。随着 .bit 越来越受欢迎，浏览器可能会对该协议提供原生支持。

大多数人不需要创建 .bit 域名，因为那会增加不必要的麻烦，让用户需要安装额外的软件或通过代理才能访问你的网站。

在名称币区块链中注册一个用户名相当容易。只需要将一些比特币换成名称币，下载钱包，然后就可以注册你的用户名了。但是如何登录到名称币区块链中呢？认证和授权的工作机制是什么样的？最近创建的 NameID（https://nameid.org）是个两全其美的方案：用户可以用自己的名称币 /id 登录到 OpenID 支持的所有网站中（数以千计）。这样，名称币终于可以无障碍地进入主流 app 市场了。

不过，将身份标识去中心化的代价是什么？好吧，其代价跟把数据用 IPFS 去中心化，把财富用比特币去中心化一样：用户必须保管好自己的私钥。对于黑客来说这没什么，他们喜欢去中心化和保护隐私。当涉及在互联网上使用正确的工具这一话题时，他们是最理想的人群。在追求所用工具的高效和完美上，黑客们会为自己天生的驱动力而自豪。他们用 GPG 加密通信，用 Tor 客户端来防御自己的浏览历史。

对他们来说，为了去中心化额外存储些私钥完全没问题。但主流人群呢？他们真的关心这些吗？我觉得他们并不是特别关心隐私和去中心化。再考虑到计算机安全知识的平均水准，公平地说，我觉得大多数人都不能或者不愿意安全地保管加密密钥。迄今为止最大的比特币应用 Coinbase（https://coinbase.com）在市场上的成功就是明证。Coinbase 跟去中心化是对立的：它是比特币银行。它提供了保管私钥的服务。

很多比特币社区反对任何形式的集中，一些人甚至连 BitTorrent 追踪器这种轻微的集中都不想碰。真正的问题是：你愿意在多大程度上去中心化你的软件？想去中心化域名并让用户保管三组不同的密钥吗？这个问题的答案取决于你的受众群，以及去中心化带来的收益是否值得你这么做。比如提到“去中心化的 Dropbox”这个 Dropbox 的竞争对手时，答案可能是肯定

的。如果有竞争对手能承诺在保证安全的前提下将数据去中心化，我敢打赌，肯定有足够多的人认为有充足的理由来安全地保管好一个私钥，以便让这样的系统工作。

即便那些人真的不想保管，也会有商家跟进，提供存储即服务的业务。必须承认，即便用了这么长时间的比特币，我也还是在用 Coinbase 的服务保管我的比特币。我只是不想因为持有比特币而提心吊胆地害怕自己的电脑被黑！我之所以更相信 Coinbase，是因为他们持有很多用户的资产，CEO 看起来也值得信任（至少比 Mt.Gox 的 Mark Karpelès 可信[4]），并且其背后的两个投资商（安德森·霍洛维茨和联合广场风险投资公司）也很可靠。

我认为，我们不需要创建完全没有信任关系的系统，而是应该创建更可信的系统。我喜欢一个关于火车的例子。假设有一列从旧金山开往洛杉矶的火车突然撞车了。如果火车的控制权集中在调度员手里，我们就知道谁应当承担责任（调度员）。如果火车的控制权分散到了每个乘客手里，则无法追究每个人的责任，也很难找到那个坏人。

去中心化本身没什么好的，必须在明确的目的下和真实的用例中才能体现其优势。dapp 可以有不同的去中心化程度，这要取决于它们各自的用例。毫无疑问，如果要创建一个组织秘密活动的应用，那就应该是去中心化的 dapp。如果要创建一个想得到广泛认可的社交网络，用 .bit 做域名很可能并不是什么好主意。

如果你用了一个把数据存储在 IPFS 上的 dapp，并且它还用比特币区块链上的彩币发行了自有“货币”，那么你很可能也会用 NameID 存储用户的身份标识。这里可能会有三组密钥合并到某种本地或者第三方的密钥库中，让用户可以访问并使用你的软件。

2.4　去中心化计算

我们已经介绍了数据、财富和身份标识的去中心化存储，那么计算呢？我们能把 Web 应用程序直接存放在 IPFS 上并运行它吗？好吧，既可以也不

注 4：Mt.Gox 曾经是全球最大的比特币交易平台，其创始人兼前首席执行官 Mark Karpelès 因涉嫌操纵公司计算机系统进行财务造假，于 2015 年 8 月 1 日在日本被捕。——译者注

可以。IPFS 只是一个文件系统，跟其他所有的文件系统一样，在它上面运行和显示静态网站完全没问题。但对于我们现在称为后台系统的动态应用程序而言，则需要一个编译和运行环境，比如 Node.js 和 Ruby on Rails——IPFS 不行。因此，虽然可以把应用程序的数据存放在 IPFS 上，也还要考虑要把源码放在什么地方。

为此，我们有两个选择。第一是把数据保存在 IPFS 上，把源码托管在传统的虚拟机（VM）提供商那里，比如 Heroku。VM 模拟了特定的计算机系统，其操作是基于（假想或真实的）计算机功能和架构的。VM 的实现可能涉及特殊的软件、硬件及两者的组合。Heroku 是非常流行的平台即服务（PaaS）提供商，让用户非常轻松就能用上 VM。虚拟机上可以运行 Go 和 Node.js 等代码写成的动态后台系统，还可以用 MongoDB 这样的内部托管的数据库来存储数据。

如果把源码放在 Heroku 上，数据放在 IPFS 上，那么用户仍然会相信数据是属于他们的，你没有把数据卖出去赚钱。但他们不能保证在服务器上运行的代码就是你开源的代码。除了无法验证，这还意味着有中心失效点（Heroku）。第二种办法是把数据保存在 IPFS 上，把源码部署到构建于 IPFS 之上的去中心化 VM 上。有这样的东西吗？最接近的项目是 astralboot（https://github.com/ipfs/astralboot）。这基本上是一个 golang 服务器，只是它直接从 IPFS 上拉取文件，并且允许你运行基于 IPFS 的 Debian 环境。也就是说如果你在 astralboot 上部署了一个动态应用程序，它是搭建在 IPFS 上的，你只需要在 astralboot 上的 Linux 环境中配置出特定的环境。

另一个选择是以太坊自己的 EVM（以太坊虚拟机）。以太坊的区块链跟比特币区块链有很多不同：有不同的块时间，图灵完备的合约，并且它是一个去中心化状态机。我认为它虽然是 VM，但并不完整，最起码肯定不是大多数开发人员想要的那种 VM。在今天的软件市场上，几乎一定要从第三方那里请求数据。在这一领域有很多竞争者，他们专注于数据利基市场，为你提供指向其他服务的 API。与其每次都做重复性的工作来为你的应用创建可信的数据源，还不如直接用第三方的 API。以太坊 EVM 的问题是无法获取区块链之外的数据，除非数据提供者已经在自己的服务器里设置好了智能合约，能够跟以太坊协作。

这对于作为 Oracle（可信的联合数据源）的新 API 来说是好事，但对于已有服务来说却不太妙。以太坊区块链和比特币区块链都不能从外部请求数据。这种不便是故意实行的安全限制。如果能从区块链里调用 API，黑客有可能用各种数据请求逃脱区块链，最终结果就是造成网络膨胀。所以单独将区块链当作完整的 VM 来用不是个好主意。

另外一个项目是 Go-circuit（http://gocircuit.github.io/circuit），会创建在机器集群上运行实例的小型服务进程。它们形成了一个流畅的高效弹性网络，来自任何一台机器的分布式进程都可以协调同步。这是一个为 Go 程序准备的分布式运行时，还集成了 Docker。如果你的项目用的是 Go 语言，那它很棒，否则的话它对你来说就没什么用。

所有这些关于去中心化计算的讨论都提出了一个问题：如果计算是个市场会怎么样？想象在一个网络中，真正有用的工作量证明计算是在侧链中，像网格币（https://gridcoin.us）和素数币（https://primecoin.io）之类的一些币种已经在这么做了。但它们对用户决定的新计算来说是不可用的（以可验证的方式），它们是基于造币者需要的现有计算的。我们需要的是 dapp 开发者能够通过易于访问的接口部署代码的 P2P 去中心化计算。

我们想要的网络有自己的计算币，dapp 开发人员会为了计算他们的代码而向挖矿机、计算引擎支付费用。随着用户量的增长，网络的价值会不断增长，从而带动计算币的价值跟着增长。这是加密研究的一个领域，所以我相信，肯定有像 astralboot 这样的产品在加紧研发中，它们是以最快方式做原型的希望。

如果你觉得 astralboot 太难配置，也还有 Heroku 可以用。如果 Heroku 失效了，而你的代码是开源的，那么任何人都可以把它重新上传到新的服务器上，访问那些放在 IPFS 上永久的、用户拥有的、可公开验证的数据。如果计算市场成为现实，你就可以直接从浏览器上运行动态应用程序了，就像现在运行在 Web 的域主机上一样。

2.5 去中心化带宽

之前已经讨论过 dapp 中能去中心化的 4 个重点了：数据、财富、身份标识和计算。我们还谈到了域名注册，不过在大多数情况下都没有这个必

要。接下来我们要讨论去中心化带宽。

对于大多数用户而言，ISP 在你和互联网之间充当网关的角色（见图 2-4）。有些 ISP 跨越国家，作为中央中枢把人们连接起来。此外，ISP 还解决了“最后一公里”问题，把终端用户连接到更快、容量更大的互联网“主干网”上。“最后一公里”只是电信行业用来指代电信网络最后一层分支的词语，它将通信连接送到散客那里。真正直达客户的是互联网网线。

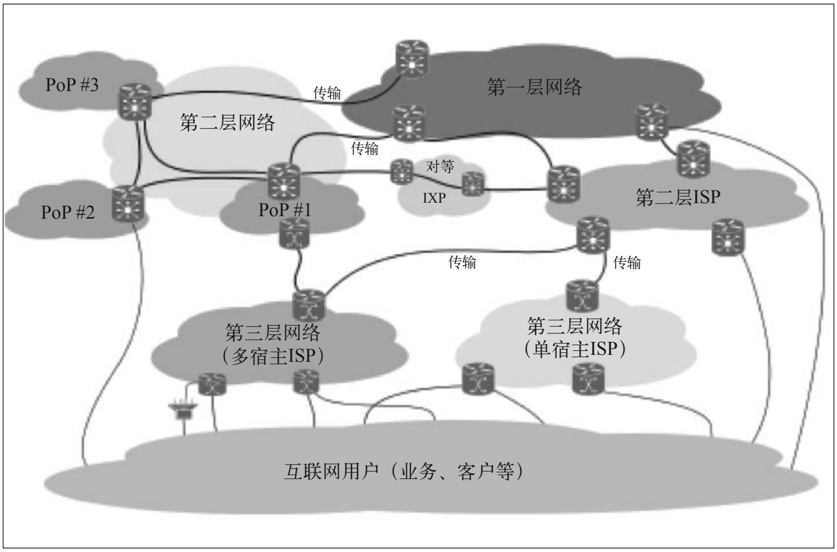


图 2-4：互联网

AT&T 和 Comcast 之类的 ISP 靠为我们提供互联网接入来获利，因为现在除了 ISP，我们没有别的选择。其缺点是，这些集中式网关也是中心失效点。政府可以根据需要关掉它们。ISP 还必须遵守当地的法律，创建不允许用户访问的 IP 黑名单。

这全都管用，因为现在还没有什么能取代它，不过有其他选择已经开始出现了。最新的例子是 iOS 上的 FireChat（https://opengarden.com），这款应用是由一个叫作开放花园的公司创建的。FireChat 利用 iOS 的多端连接特性，实现手机之间点对点的直接对话。它不需要 ISP。FireChat 是**网状网络**应用程序的范例。网状网络是标准的集中式互联网的去中心化版。在网状网络

中，用户不需要通过中心网关来访问站点。他们可以直接连接到最近的路由器上，一般就是附近的电脑。

现在有很多已经投入使用的网状网络。西班牙有世界上最大的网状网络之一，那里有超过 50 000 人需要访问互联网，却没有 ISP 为他们提供网络接入。在纽约遭遇飓风期间，网线断掉的情况下，网状网络被用于传递宝贵的救援信息。在旧金山也有大量的“暗”网，只对其所创建的秘密社团内部的访客开放。网状网络一般无法访问常规互联网及其中的数据。如果环中没有隧道，就无法访问到正常的网状网络。只有通过隧道到网状网络的来回切换才能建立起隧道。现在还没有主流硬件厂商能够支持这两项工作同时进行，但你可以借助开放花园和 CJDNS（https://github.com/cjdelisle/cjdns）这样的项目来使用混合网络。这要稍微难一点，因为除了软件，还涉及硬件的变更。路由器应该有能力同时访问两个网络，这样就能从常规互联网中拉取数据并用在网状网络中，从而让它不能被关掉。

尽管去中心化带宽值得拥有，但只有在互联网审查和 Web 访问被阻断的情况下才有这个必要。我觉得如果大范围出现这种情况，现实需求将会把去中心化带宽变成主流选择。使用区块链，我们能让其他计算设备取代 ISP 成为网关。他们能通过用“加密货币”路由数据和带宽证明来获取报酬。

就像“加密货币”能将计算和数据存储的 P2P 市场变成现实一样，“加密货币”也能促成带宽分享。“加密货币”能在集中式力量存在的地方培育出市场。这些市场可能是计算、存储、带宽，以及任何能想象到的“真实”和“人造”的稀缺资源。我们将会看到，经济变得越来越依赖于信息。随着所有基于劳动力的事情慢慢被自动化吞噬，数据市场极有可能慢慢变成最大的市场。

2.6 去中心化资产的去中心化市场

谈到市场，随之而来的还有衍生品、资产、货币和期货等金融工具。一个问题会浮现出来：人们在哪里交换这些资产？一直以来，我们都是用集中式的证券交易所来交换这些资产，但在去中心化的信息经济环境下，该怎么办？在出现更多的基础设施之前，资产交易一般都是去中心化的，比如在以货易货的经济时期。现在，我们可以把互联网提供的开放式信息交换

和所有权验证的去中心化模型结合起来。在这种情况下，也可以把货物看作没有任何中央监管的资产。政府提供了可信的中介，确保跨境交易的合法性和价值的稳定性。在交易和资产的保护上需要投入很大的信任。到目前为止，政府是我们所能创造的最强信任源。

由政府创建的资产（比如国家货币）一直处于政府严格的监督之下。那么，让非监管下的玩家参与进来会怎么样？现在，针对比特币网络以及该如何监管而展开的调查有很多。对于一个不受节制的、自由的国际市场来说，需要的是一个适合新经济的去中心化证券交易所，不依赖任何实体的监管。在以 dapp 为主流的信息经济时代，dapp 的每个用户都是股东。

他们拥有的股份是 dapp 的自有“货币”、彩币或者很有可能是比特币的侧链。这些“货币”的估值会跟随 dapp 的估值而波动：觉得这个 dapp 有用的人越多，它的估值就越高，那么它的股价自然也会变高。此外，这些股份还能从 dapp 的收入流中获得股息，这进一步提高了其价值。用户最终想要将这些资产换成更加持久和稳定的资产，可能是有美元支撑的比特币侧链，比如 BitUSD。 BitUSD 是一种新型的“加密货币”，与美元挂钩，可以无缝换取 BTC。也可以用其兑换价格稳定、内置铸币税的“加密货币”，或者自有去中心化预留的“货币”。简言之，用户希望资产的价值稳定。最终，当用户在对资产的价值满意并且不想冒险失去在 dapp 中的投资时，就会将它们换成能维持自己购买力的东西。

到目前为止，“加密货币”的兑换还必须靠中心机构来完成，你要把钱交给他们，所以在兑换期间，你只能相信他们。结果就出现了很多由这些中心机构进行的盗窃行为，因为很多政府仍在对比特币进行检查，所以这些中心机构大都不在法律合同的管辖之下。最臭名昭著的就是 Mt.Gox，这家位于东京的交易所由 Mark Karpelès 经营。在比特币的初期阶段，Mt.Gox 有很多用户，规模非常大。它在 2014 年 2 月关闭，没有对客户资金的损失做出任何解释。人们简直被气疯了，但由于关于比特币的法律很模糊，所以几乎没办法惩戒 Karpelès。

从那以后，人们越来越不信任交易所。在理想情况下，我们可以把 dapp 的资产 / 股票价格放到政府的证券交易所的交易列表里去，但那样需要经过很多的审核批准环节，人们可不想等。另外，一家公司要想进入联邦证券交

易所还需要进行 IPO。IPO 要求这家公司有上百万的资金，雇用投资银行家和律师，还要做成堆成堆的文件。这个进入门槛非常高。因为证券交易所是国家指定的垄断性贸易机构，所以他们能够维护那样的系统。

比特币给了我们机会，让我们可以选择退出金融系统，而这也是我们应有的选择。我们可以把 IPO 去中心化，让早期创业者把他们的股票卖给没有经过认证的投资者；而按照投资者认证的要求，只允许人们投资那些银行里至少有 100 万美元存款的公司。众筹是个好办法，但它给投资者的回报是不公平的。我们需要一个去中心化的证券交易所。那么，它该怎么运转呢？

有些想法跟瑞波币（https://ripple.com）风格的兑换机制有关。（瑞波币风格允许你选择跟谁达成共识，而不是像工作量证明那样相信大多数挖矿机的工作。）恒星币（https://stellar.org）从某种角度来看是新的瑞波币，其创始人 Jed McCaleb 跟 David Mazieres（斯坦福大学分布式系统研究方面的天才）等 Stripe-backed 的队友们组队，决定创建一种新的代链。恒星币的源码跟瑞波币的源码基本上一样，只是要换一种新打造的共识机制。

其基本思想是，只要你相信一个管理交易的第三方中介机构，就可以跟其他任何人交易你的货币。可信的第三方可能是银行或中心交易所，甚至只是一个朋友。这个系统很有意思，因为它不是完全去中心化的，跟大多数代链都不一样。它增加了一个中间人或中间组织，借助社交信任来辅助完成交易。

这确实不够理想，但完全去中心化的零信任订单匹配还没有解决办法，并且在 AI 得到跨越式发展之前几乎是不可能的。集中式有它的缺点，但也有积极的一面：可问责。在恒星币的模型中，中介节点要为所有事故负责。如果真的出了事故，他们的名誉就会受损。这个模型确实有它的价值，但就跟大多数山寨币一样，恒星币的问题在于引入了一种新的“加密货币”，而且它的共识机制还有待验证。

工作量证明是唯一一种可以跟大规模女巫攻击对抗的共识机制。留给我们的是什么呢？据我所知，去中心化程度最大的交易所是 Mercury（http://mercuryex.com），是一个名叫 Mappum 的开发人员创建的。Mercury 是一个多币种钱包，用比特币的**交叉原子链**（CAC）传输协议来兑换各种“加密

货币”。CAC 传输协议能让拥有不同“加密货币”的甲和乙进行兑换，不需要可信的第三方。前提是两种“加密货币”都需要实现这一协议才行。真正的订单匹配是在服务器上完成的，但价值在钱包本地保存着，所以不会有被偷的风险。Mercury 是一个在 GitHub 上的开源项目[5]，还在开发当中，但已经有了一些用户。随着这个钱包渐渐流行，用户会想把他们的资产都放在里面，点击一个按钮就可以在不同形态的价值之间进行互换。它是圣杯钱包的先驱，也就是一个保存所有“加密货币”的钱包。

很可能不会有这样一个能识别各种竞争性协议的钱包，就像 Web 浏览器不能识别“上都计划”或其他任何 HTTP 的手下败将一样，然而所有人依然用浏览器访问 Web。比特币、彩币和侧链会最终胜出[6]。基于 BTC 区块链的资产是相对安全的，因为有超过 500 个超级计算机的计算力来保证比特币区块链的安全，它在市场份额和认知程度上具有先发优势，以及杰出的开发者社区还在扩大它的范围。任何一家公司都可以把他们的“货币”添加到开源的 Mercury 钱包里，不需要 IPO。

即便 Mercury 赢不了，这个模型也很好，它意味着那些想通过“加密货币”发行资产的人不需要跨过 IPO 这样的障碍了。

2.7　务实的去中心化

要想在现代化的世界里取得成功，必须遵守政府法规。看看 PayPal 和 Coinbase 就知道了，政府合规性可以建立起用户的信任，让业务得到国际化发展。不幸的是，作为一个已注册的公司，要在去中心化证券交易所向未经过认可的投资者发行资产，简直就是公然对抗政府法规。那么开发者该怎么创建他们的 dapp，才能让其跟 Facebook、Instagram 和 Vine 一样变成主流应用呢？这里有三个建议：

- 创建非盈利性的公司；
- 在法律文书中，将资产作为解开功能特性的应用代币；
- 将你的资产放到 Mercury 这样的去中心化交易所里。

注 5：地址为 https://github.com/mappum/mercury。——译者注

注 6：本书仅代表作者个人观点。——编者注

这是一个务实的开始，可能也是有效去中心化所必须做的全部事情。这样你就可以把去中心化交易在法律上的雷区全都交给交易者自己去处理了。交易是去中心化的，所以无法关掉，不过订单匹配服务器可逃不掉。但因为是开源的，所以用户总能搭建新的服务器，或插入他们的地址。

用侧链和彩币可以消除因处理各种不同区块链而带来的压力。它们有些支持 CAC 交换，有些不支持，需要通过给代码打补丁来实现兼容。如果底层都是比特币，那么对程序员来说，在搭建应用时使用几种不同的“货币”就没那么困难了。比如说，假设有一个搭在 IPFS 上的去中心化 Dropbox。用户会支付底层网络费用（“文件币”），可能还有在名称币区块链上注册用户名的费用，以及使用服务支付的一些“内部货币”。一条交易怎么分拆出三种不同的“货币”，从而命中不同的区块链？如果它们底层用的都是比特币，那价值就可以在不同币种间无缝转移。

因此，比特币是其他所有构建于区块链之上的金融资产的最强基础：它比其他“加密货币”都好用，不用太费心思考，实现起来也简单。但在币种和用例之间，该如何抉择呢？比特币的交易时间可以更快，作为侧链的莱特币可以把速度提高很多。还有暗币，可以对区块链上的交易数据进行加扰和加密，从而保护交易历史的隐私。素数币和网格币让你利用工作证明解决科学问题，不用把计算力浪费在支持网络上。怎么才能从各种“加密货币”中挑选出我们想要的功能，而不是把它们全都放在一起，然后从中挑选“加密货币”呢？

答案藏在所有“加密货币”的统一封装器里。可以做一个存储所有基于比特币的“加密货币”的钱包，就像 Mercury 想要做的那样。这很实用，不过还是要让用户为在什么时候花哪种“货币”而烦心。相反，用户应该只能看到一种“货币”的余额和功能，并且可以根据自己的意愿开关。封装器应该提供可以将各种“货币”轻松切换的功能。

在后台的安装应该像 Node.js 的包管理那么简单。想让交易变得更快？`bpm install lite`。私密交易？`bpm install dark`。帮助计算科学数据？`bpm install solar`。交易应该经过各种必要的区块链的过滤，以确保它们有用户请求的所有功能。用户可以分开存储各种资产，就像在他们的投资组合账号 / 在线钱包或现金钱包中一样，做一个一体化财富管理钱包。

让我们来考虑一些实际问题：把钱存在电脑上的钱包里是危险的。这是银行之外的去中心化选择。有些人觉得值得冒这个险，因为这是一个有思想的决定。有些人不喜欢银行，因为银行有可能破产——既然是集中式机构，就有可能失效，它是把很多人的“蛋”放在一个“篮子”里。但大多数人不是那么有思想！现实就是这样，大多数人不关心集中式、数据安全或数据所有权。他们想要的只是一些能顺畅地帮他们解决问题的东西。

银行帮人们解决了一个非常重要的问题：财富的安全保管。保管作为一种服务已经存在了相当长的一段时间，并且仍将继续发挥它的价值。比如 Coinbase，它是比特币领域最受欢迎的服务，但跟去中心化完全沾不上边。它基本上就是一家银行：不仅在一个安全的线上钱包中持有你的比特币，而且开始持有你的美元了。随着 Coinbase 日渐流行，比特币也变得更受欢迎了。它很好用，是个让人放心的保管处所。“如果比特币成为主流，银行也要把它当成货币一样予以接受”。这听起来挺荒谬的，因为中本聪创建的比特币是为了提供了一种可能，让我们避免遇到现在所有在线交易都有的拒付问题。我们为什么要把它存到银行里呢？这个区别类似于随身携带美元还是银行卡。

如果我们像使用现金一样使用比特币，在 Web 上就不会碰到拒付交易。但大部分人会选择使用类似于银行的服务。银行运转良好时提供的服务很棒。我们可以在任何地方用钱，不用担心它会被偷走，这都是因为银行对我们的账户进行了保险，并且能安全地保管我们的资产。传统的银行可能也会升级，以便接受比特币的转账，否则就可能被 Coinbase 这样新的竞争者超越。想象一下吧，银行里有你的比特币账户。他们会用你的本国货币来显示余额，还会有关联的账户号码、路由号码和比特币公钥对。

如果有人给你发比特币，它会直接到你的银行账户里。如果有人给你发政府货币，它也会直接到你的银行账户里。因为有通用封装器，你应该可以将价格稳定性作为一个功能打开，所以你的账户应该可以像现在这么稳定。你可以像使用政府货币那样随意使用比特币，因为它们将会变得难以区分。

比特币会找到它的利基市场。大多数人不会关心比特币这个词，而只是想用他们自己的钱。像 Abra（https://goabra.com）这样的比特币领域领导者甚

至根本不提比特币这个词。他们只会把它作为快速减免国际汇款和微支付的一种协议。

Abra 是一家创业公司，瞄准了发展中国家汇款这个巨大的市场，它要借助一系列去中心化的柜员来提供政府货币和比特币之间的相互兑换。他们没有用比特币这个词，而是用了“数字货币”。这样做很聪明，不会吓跑主流客户，还很有可能会促进他们的销售。在硅谷之外，人们真的不关心比特币，这是个很现实的问题。

比特币 ATM 的兴起确实是个令人着迷的想法，但如果能去那些没有恰当的金融基础设施的发展中国家看看，你就会明白，我们这辈子是不可能看到传统的 ATM 网络全都转变成比特币 ATM 了。与此相反，比特币应该在需要时（电汇，特别是国际电汇）变得更快、更便宜，从而成为现有支付设施的补充；并且要将“加密货币”能提供给用户的功能全都放开，比如微支付，这可以反过来启动像微博市场这样的 dapp。

没有人是被迫使用银行、身份标识、数据或集中式计算的，他们只是为了让生活变得更轻松。在必要时，我们可以选择去中心化，并且希望用户能意识到数据的价值，希望这个世界将会慢慢懂得安全保管自己密钥的重要性。

第 3 章

创建你的第一个dapp

讲了这么多理论知识，该动手实践了。我觉得你以前应该至少做过一个软件。创建 dapp 并不比创建常规的应用程序困难多少，只是创建 dapp 要用去中心化的方式去思考，并且没有那么多成熟的类库可用。

本章会带你创建一个去中心化 Twitter，我们将会介绍：

- Go 语言
- 去中心化架构和 IPFS 分布式数据存储
- Kerala，一个 IPFS 接口
- Coinprism，一种管理彩币的钱包服务
- Mikro，一个带有内部经济体系的去中心化消息应用程序

3.1 Go语言

我们将使用 Go 语言来创建 dapp。后端开发人员对 Go 语言表现出了浓厚的兴趣，因为它没有“回调地狱”式的语法，计算速度快，还有对并发友善的“go 例程”。Erlang 和 Rust 都号称比 Go 厉害；在某些方面可能确实如此，但跟 Go 语言比，它们的类库都还非常不成熟。

JavaScript 最近也很流行。在 Node.js 问世后，JavaScript 不再是只能用在前端的语言。开发人员能用这一种语言创建并维护整个网站（当然，还要有

HTML/CSS）。JavaScript 是 Web 语言，并且 JavaScript 开发人员可以用各种各样的框架来搭建他们的 Web 应用。尽管 JavaScript 很棒，但也有它的弱点。用 JavaScript 实现并发很难，它的值构造器也不好懂。Go 弥补了这些不足，是专为分布式系统设计的编程语言。

我开发的 Web 应用既有用 Go 语言写的，也有用 JavaScript 写的。两种语言都各有优缺点，但我必须承认，用 Go 来做 dapp 的效率更高。Google 创建 Go 语言的目的就是为了满足自己在大型数据集上进行快速高效的大规模并发计算的需要，Go 很好地解决了这个问题。自从第一版发布以来，Google 内部使用 Go 语言的人数增长非常迅速。

Go 既有 C 的编译速度和能力，又有 Ruby 的简洁优雅。Go 是专为开发分布式系统打造的，所以我才想到用它来开发 dapp。另外，IPFS 也是用 Go 开发的，这也是一个加分项，因为我们将分布式文件存储集成到系统中时不会有兼容性障碍。基于 Go 的 Web 框架很多，我们的选择包括 Martini、Goji、Gorilla，甚至 Go 的标准 net/http 包。我一般会尽可能保持依赖栈的轻便性，所以选 net/http，我的 go-to，以及其他非用不可的 Web 应用库。

3.1.1 集中式架构

在构建基于服务器 – 客户端的标准 Web 应用时，有三种常用的范式，下面稍微介绍一下。

1. REST

服务器 – 客户端模型相当简单，并且已经成为在 Web 上交换数据的主要方式了。REST 指的是表述性状态转移（Representational State Transfer），是一组指南和最佳实践，用于创建基于服务器 – 客户端模型的可伸缩 Web 应用。REST 本身不是一项技术，它跟 AJAX 一样，是一种实践。这种实践鼓励大家使用 HTTP 协议中早已存在但极少使用的各种能力。用户只是将浏览器指向一个 URL（统一资源定位器）就会发送一个 HTTP 请求。每个 HTTP 请求中都会包含一些参数，服务器可以根据这些信息来决定给发起请求的客户端什么样的响应。

2. CRUD

CRUD 指的是创建 – 读取 – 更新 – 删除（Create-Read-Update-Delete）。这些

都是数据存储库上的基本操作，用来直接处理记录或数据对象。离开了这些操作，记录就仅仅是被动的实体。通常情况下，它们就只是数据库表和记录。REST 跟运行着的系统进行交互，而 CRUD 处理系统中的数据。开发人员一般会使用 MongoDB 或 MySQL 之类的数据库在他们的数据上执行 CRUD 动作。

3. MVC

MVC 指的是模型－视图－控制器（Model-View-Controller），是目前最流行的软件编程范式。模型管理着应用程序的核心行为和数据。视图渲染应用程序的用户界面。控制器接收用户输入，根据需要来调用模型对象和视图，从而执行特定的动作。

3.1.2　去中心化架构：IPFS介绍

那么，在一个去中心化的架构中，CRUD 和 REST 会变成什么样呢？它们会合而为一。因为就 IPFS 而言，数据会分散到去中心化网络中，而不是由某个人控制的计算机上。在本地执行操作或处理请求跟在远程是一样的。你自己和其他人都既是服务器，又是客户端。这听上去挺复杂，但实际上并非如此。我之所以选择 IPFS 来做去中心化数据存储，就是因为它远远超越了该领域中的所有竞争对手，有经过多年研究积累的优秀思想和经过验证的实战经验。

dapp 不是运行在服务器上的，而是运行在每个用户自己的计算机上。我们还没实现去中心化计算。将计算上传到一个集中式的虚拟机上，比如 Heroku，会违背去中心化的初衷，所以正确的做法是将 dapp 作为一个可下载的二进制文件发布。用户可以将 dapp 下载到自己的桌面上，然后用 Web 浏览器来访问它；也可以直接运行在一个客户端界面里，比如 Spotify 或 Skype。

dapp 需要某种形式的数据存储，因此相当于两个 IPFS 分布式文件存储节点。另一种选择是只用第三方服务器上的 IPFS 节点来存储数据，但那样的话这个云服务提供商就会变成中心失效点。毫无疑问，肯定会有人购买 Amazon EC2 的空间来放置这样的节点，以便给初学者提供 IPFS 节点服务来让他们更容易上手。当有人逐个地请求文件时，数据会从那里复制出来。对于移动 dapp 而言，IPFS 云节点也是个不错的选择，因为运行 IPFS 节点

需要占用大量的处理能力，所以会缩短移动设备的电池使用时间。

促使人们提供节点的动机是上传数据的用户会得到法定货币或“加密货币”。IPFS 的缔造者 Juan Benet 发表的一篇论文[1]介绍了一种叫作“文件币”的“加密货币”。虽然它可以满足这一要求，但可惜的是还没开始实现。与此同时，任何人都有机会创建一种激励机制，来为 IPFS 的数据存储提供激励。这样节点不需要上线就能使数据可用。去中心化程度越高越好。即便某个 IPFS 节点服务器挂掉了，如果数据还有用，那么每个请求过它的人都会有一份副本。IPFS 太美了，所以它的缔造者才称其为恒久的 Web。你也可以付费让服务器“钉住”你的数据。现在可能没有人需要你的数据，但他们早晚会需要的。这样，只要**有人**需要，数据就还在。

做一个移动应用应该挺棒的，但对于这个演示教程，我准备把重点放在编写桌面端 dapp 上，因为 IPFS 还没有非常可靠的 Swift/Objective-C 或 Android 封装。

我们看一下 IPFS 中的两个关键命令：

- `ADD`，向 IPFS 添加数据
- `CAT`，从 IPFS 读取数据

注意到了吗？这里没有删除命令。IPFS 是恒久 Web！ 在你把数据添加到网络中以后，除非你是唯一一个存储数据的人，否则根本没办法删除你添加过的数据。这是因为其他访问过该数据的节点马上就有了一份副本。另外还要注意，这里也没有更新命令，因为 IPFS 内置了 Git 的方法。在你更新一个文件时，文件本身并不是被删除，而是被版本化了。你可以给那个文件创建一个 merkleDAG，让最新的散列值对应的就是最新版的文件。所有旧版文件都还在，如果想要的话，你仍然能找到它们。

在向 IPFS 添加数据时，本质上是在网络上发送了一个广播，告诉大家你有这份数据。实际上，你并没有把数据发送给任何一台计算机。数据只在有人请求时才会发送。另外，因为数据放在网络上，所以命令的处理结果也是发生在网络上的。

注 1：http://filecoin.io/filecoin.pdf

通过修改名称，IPNS（IPFS 上的名称层）实现了表面上的更新和删除。你可以用 IPNS 在不可变的同伴 ID 下发布一个数据的 DAG。这样，当有人解析你的同伴 ID 时，他就能得到 DAG 散列值。因为 IPNS 只能给每个同伴 ID 保存一个 DAG，所以如果你想更新或删除数据，只要重新发布一个 DAG 到你的同伴 ID 上就行了。后面会介绍实现细节。

那 MVC 架构呢？

哦，它还在。什么？不是用高度创新的方法来组织我的代码吗？不是，模型还是一样，控制器用 IPFS 进行数据存储和获取，视图仍然是 HTML/CSS/JavaScript。

智能合约呢？它们扮演什么角色？

在 dapp 中，有些元素需要通过智能合约达成共识，这通常需要一台服务器。这方面的好例子不仅有用户名，还有托管和资产所有权等金融行为。从技术角度讲，智能合约就是“模型”，你可以通过交易给它们输送数据，但它们不是 MVC 架构中的主流“模型”。它们能跟已有的模型一起工作，但真的只适用于特定场景。这些案例会逐个出现，后面还会介绍如何构建智能合约。俗话说，我们需要聪明的模型、削瘦的控制器和无脑的视图。

Eris Industries 有个搭建 dapp 的框架，叫 Decerver。在它的网站上有很多文献，解释了其用法以及它为了让创建 dapp 更容易而实现的所有与众不同的革命性方法。它说模型是智能合约，但问题是智能合约是付费即可用的，并且应该不与模型的创造相关。这是种没必要的复杂性。MVC 仍然适用于去中心化应用程序，只是控制器不是跟服务器通话，而是跟区块链和 DHT 通话。

3.2 我们要创建什么

我们准备创建一个去中心化的 Twitter 作为第一个 dapp。IPFS 的 bitswap 机制意味着所有最近的节点都可以从本地的节点上拉取数据。去中心化 Twitter 很实用，之前也有人做过。一个名叫 Miguel Freitas 的巴西开发者几年前创建了一个 Twitter dapp，名为 Twister。可惜，Twister 被垃圾制造者掌握的各种安全漏洞所困扰，Freitas 只能用他仅有的工具做粗陋的修复。

补丁之所以粗糙，是因为他们采用的技术是让注册用户完成工作证明来验证其身份标识，而这是阻止女巫攻击用的。结果，这变成了新用户不愿跨越的门槛，几乎没人愿意为了试一下这个系统而奉献计算力来证明自己是好人。Twister 的安装和配置也很困难。

新版的 Twitter dapp 很合我们的胃口，所以我们准备用 IPFS 和比特币这些新技术做一个。我们给它起的名字是 Mikro。把它作为第一个 dapp 非常棒，因为它就像是 dapp 中的 MVP。数据相当简单直接：你是用户，并且会输出消息（micropost）。你可以发现新的用户，并查看他们的消息。

3.2.1 配置

我们先来配置 Go 的开发环境。我是极简主义者，不会增加无谓的复杂性。Go 有 Linux（https://storage.googleapis.com/golang/go1.4.2.linux-amd64.tar.gz）和 Mac OS X（https://storage.googleapis.com/golang/go1.4.2.darwin-amd64-osx10.8.pkg）下的安装包。（Windows 用户们，抱歉了，我们把重点放在了类 Unix 系统上。）

这些安装包会自动将 Go 安装到 usr/local/go 下，并设置好环境变量 `path`。环境变量 `path` 是软件配置中的“陷阱”之一。它将你的库链接到终端关键字上，方便调用。如果它没有设置 `path`，可以像下面这样手动设置：

```
export GOROOT=$HOME/go
export PATH=$PATH:$GOROOT/bin
```

在这个例子中，`$HOME` 是我们安装 Go 的路径（usr/local/）。

装好之后，我们要测试一下，以确保一切正常。在 src/ 文件夹下，创建一个名为 tests/ 的新文件夹，然后在里面创建一个 helloworld.go 文件。在终端中输入下面的命令，开始编辑这个文件：

```
nano helloworld.go
```

将下面的代码片段添加到文件中，然后保存：

```
package main
import "fmt"
func main()
```

```
{
    fmt.Printf("hello, world\n")
}
```

接着用 Go 运行它：

```
$ go run helloworld.go
```

如果控制台中显示了 `hello，world`，说明 Go 安装成功了。

太棒了！现在可以安装依赖项了。首先要安装的是 IPFS。Go 都是直接从 Web 上的源安装依赖项。在控制台输入下面的命令就可以安装 IPFS：

```
go get -u github.com/ipfs/go-ipfs
```

安装完成之后，运行点命令执行刚修改的初始化文件，使之立即生效：

```
source ~/.bashrc
```

`go get` 命令会把依赖项下载下来并构建好。它们会被存放在 Go 根目录下的 src 文件夹中。如果你用命令 `cd` 进入 src 文件夹，会发现一个 github.com 文件夹。Go 会把库的 URL 分割开，用其中的每个组成部分创建一个文件夹。因此在 github.com 文件夹下面，会有一个 ipfs 文件夹。它下面还会有一个 go-ipfs 文件夹，以此类推。这很有用，因为我们会从一个源头拉取很多依赖项，而 Go 会自动把它们放到各自的文件夹中。因此，所有来自 GitHub 的依赖项都会放到 github.com 文件夹下，然后是 URL 路径对应的各层文件夹。

要开始使用 IPFS，还需要在系统中初始化它的配置文件，执行下面这条命令：

```
ipfs init
```

这需要花几秒钟的时间。它会把自举（硬编码）同伴添加到你的配置中，并且给你的节点一个身份标识密钥对，以便在你添加或固定文件时向网络标识你的同伴身份。

在初始化完成之后，在终端中输入 ipfs 应该得到下面这样的提示：

```
ipfs - global p2p merkle-dag filesystem
ipfs [<flags>] <command> [<arg>] ...
```

```
 Basic commands:

 init      Initialize ipfs local configuration
 add <path>Add an object to ipfs
 cat <ref> Show ipfs object data
 ls <ref>      List links from an object

Tool commands:

config    Manage configuration
update    Download and apply go-ipfs updates
version   Show ipfs version information
commands  List all available commands
id            Show info about ipfs peers

Advanced Commands:
daemon    Start a long-running daemon process
mount     Mount an ipfs read-only mountpoint
serve     Serve an interface to ipfs
diag      Print diagnostics

Plumbing commands:
block     Interact with raw blocks in the datastore     object
Interact with raw dag nodes  Use 'ipfs <command> --help' to
learn more about each command.
```

这些就是 IPFS 的所有命令，看到这些就表示你的安装成功了。现在试着添加些东西到 IPFS 上去：

```
ipfs add helloworld.go
```

它应该返回类似下面这样的东西：

```
# QmT78zSuBmuS4z925WZfrqQ1qHaJ56DQaTfyMUF7F8ff5o
```

这是你刚刚所添加数据的散列值。数据还在你的计算机上，但它现在有了一个与之关联的内容地址。只要有这个地址，任何人都可以在你在线时直接从你的计算机上获取这个文件。一旦得到了这个文件，他也就有了这份数据。需要这份数据的人可以同时从你和他的计算机上按比特获取数据。存储数据的同伴越多，下载速度越快，就像 BitTorrent 一样。但 IPFS 比 BitTorrent 更棒，它还内置了版本和命名系统。

既然 IPFS 上已经有你添加的数据了，我们试着把它拿回来：

```
ipfs cat <that hash>
```

这个命令会把 helloworld.go 拉过来并显示在控制台中。现在它是直接从你的计算机上拉取的。

下一个依赖项是 Kerala。Kerala 是我写的一个小封装器，封装了 IPFS 和彩币，可以帮我们创建去中心化 Twitter。不过 Kerala 并不是专用的，也可以用来创建其他的 dapp。有了 Kerala，从 MerkleDAG 向 IPFS 添加数据就更容易了。你可以在终端里执行下面这个命令安装它：

```
go get -u github.com/llSourcell/kerala
```

下面这个例子说明了用它向 IPFS 中添加和获取数据有多容易：

```
//启动节点

node, err := kerala.StartNode()
if err != nil
{
panic(err)
}

//将你的文本添加到IPFS(创建MerkleDAG)

var userInput = r.Form["sometext"]
Key, err := kerala.AddString(node, userInput[0])

//从IPFS上获取你的所有文本(获取MerkleDAG)

tweetArray, _ := kerala.GetStrings(node)
```

代码的第一段启动了一个节点，所以你的 dapp 成了一个 IPFS 客户端。它启动了一个守护进程，向网络广播你是一个同伴。第二段向 IPFS 上添加文本。你可以向 IPFS 中添加各种类型的数据：视频、图片、数据结构。我们在这个例子中用 `AddString` 方法添加了一个字符串。你每次添加字符串，封装器都会为那个字符串创建一个新的散列值，然后将这个散列值链接到前面的散列值上。链表是个抽象术语，它基本上是说如果你请求了最新字符串的散列值，它会返回所有链在一起的字符串的散列值。

链表来自 IPFS 标记为 MerkleDAG 的数据结构。它是一个有向无环树图，可以用来关联数据。对于 Twitter dapp 来说，这很有用。你每次发消息时，封装器都会把它链到之前的散列值上，并且在本地一个名为 output.html 的文本文件中存储新的散列值。只有你知道那个散列值的密钥，也只有你能够访问该数据，但你愿意跟网络上的其他人分享。

最后一段代码基本上是在跟 peerID 关联的散列值上执行了一个“ipfs cat”（用 IPNS），并把它存在一个数组中，以便在视图中显示。

我们还需要一个名为 `httprouter` 的轻量依赖项，它可以让开发 Web 应用程序更加轻松。在终端中输入下面的命令安装它：

```
go get -u github.com/julienschmidt/httprouter
```

现在所有的依赖项都装好了，可以去下载我们准备构建的 dapp 的源码了。我已经事先把代码写好了。让你一下子从头开始写那么多代码不太现实，所以我觉得你最好先下载、编译并运行一下源码，然后我再详细讲解。在控制台中输入下面这个命令：

```
go get -u github.com/llSourcell/dapp
```

下面是 dapp 用到的所有引入库，供你参考。除了 IPFS、Kerala 和 `httprouter`，其他全都是标准的 Go 库。

```
import
        (
                "net/http"
                "github.com/julienschmidt/httprouter"
                "github.com/ipfs/go-ipfs/"
                "path"
                "html/template"
                "fmt"
                "log"
                "github.com/llSourcell/kerala"
        )
```

进入你的 Go 工作空间中的 dapp 文件夹，然后运行 `go install .`。在同一目录下运行 `go run app.go` 来运行这个应用程序。然后用浏览器访问 localhost:8080，你应该能看到一个如图 3-1 所示的个人主页页面。



图 3-1：我的截屏

页面上不会有消息，因为你还没发过呢。（上图是我发了几条消息后的截图。）现在通过文本域发四五条消息。每次提交之后，回到首页刷新页面就能看到它们了。这个应用程序的首页就是你的个人主页，它显示了你的所有消息。还有一个“发现”页面，帮你找到其他用户和他们的个人主页。我们管这个演示程序叫 Mikro。

3.2.2　路由

我们先来看一下路由。这个应用程序用了一个相当轻量的路由库（httprouter）。它是在 Go 的自有包 net/http 的基础上构建的，用它实现路由非常简单。在标准 Web 应用程序中，GET 和 POST 方法经常用来跟请求或发送数据的页面相关联。在路由中也是如此，并且数据操作（IPFS CAT 和 ADD）也是伴随它们一起发生的。

在 app.go 的 main 方法中，你会看到下面这样的路由定义：

```
//[2] 定义路由

router := httprouter.New()
//路由1 首页（个人主页）
router.GET("/", TextInput(node))
```

```
//路由2 发现页面

router.GET("/discover", displayUsers(node))
//路由3 其他用户的个人主页
router.GET("/profile/:name", TextInput(node))
//路由4 向IPFS添加文本
router.POST("/textsubmitted", addTexttoIPFS(node))
//[3] 链接资源
router.ServeFiles("/resources/*filepath", http.Dir("resources"))
http.Handle("/resources/", http.StripPrefix("/resources/",
  http.FileServer(http.Dir("resources"))))
http.Handle("/", router)
//[4] 启动服务器
fmt.Println("serving at 8080")
log.Fatal(http.ListenAndServe(":8080", router))
```

首先创建了一个结构体，初始化路由器：

```
router := httprouter.New()
```

除此之外，与大多数其他语言不同，Go 不是传统意义上的面向对象编程（OOP）语言。它遵循一种类似于 OOP 的模型，但又跟 OOP 有所区别。我们可以把结构体理解成 Go 语言中的对象，它也有数据域和方法。但在常规的 OOP 中，我们是使用关键字 `class` 定义对象的。这有助于继承，但 Go 中没有继承这个概念。尽管乍一看很不方便，但这实际上是经过深思熟虑后做出的决定。如果有很多相互扩展的类，有由多种不同的接口和实现构成的层级结构，你就知道继承能让代码变得多乱了。Go 用子类型（`is-a`）和对象组合（`has-a`）来定义结构体和接口之间的关系。

路由 1 定义的是用户访问 localhost:8080/ 时调用的方法。这是你进入应用程序后见到的第一个页面。路由 2 是发现页，发现页会显示当前网络中正在使用我们的应用程序的所有同伴。路由 3 是一个模型 URL。注意 `/profile/` 后面的关键字 `:name`，它用来加载对应用户的个人主页。当用户所访问 URL 中的 `:name` 部分出现用户 ID 时，应用程序就会加载这个用户 ID 指定的个人主页。在这种情况下，这个用户 ID 应该就是你启动 IPFS 后台程序时创建的 IPFS NodeID。每个 IPFS 节点都会有它自己的 NodeID。因为你的 Mikro 实例是个 IPFS 节点，所以你也会有一个。路由 4 向 IPFS 中添加文本。用户每次提交的消息都会通过这个路由添加到 IPFS 中。[3] 和 [4] 是配置代码，用于将服务器连接到它的资源上，并在本机的 8080 端口启动它。

3.2.3　数据存储和获取

注意 `main` 方法中最上面的“启动节点”代码：

```
node, err := kerala.StartNode()
if err != nil {
   panic(err)
}
```

这就是让你的应用程序变成 IPFS 网络的一部分所需要做的全部工作。

在你向网络中提交了 5 条消息后，就会看到它们一条条地出现在 submit text 按钮下面。你刚刚向 dapp 中添加了你的第一条数据！记住，向 IPFS 中添加数据后，并不是说它会被分成上百万条存在很多人的计算机上，也不是说不管怎样别人都关不掉它了。你所做的只是向网络发了一条广播，告诉别人你有刚才提交的数据。它还存在你的计算机上。如果你下线了，别人也就访问不到数据了。

这就是 Twister 等一些 dapp 中存在的问题：你必须一直在线。但 IPFS 就是要在关注性能的同时力争达到一个让数据永久存在的阶段。只要 Mikro dapp 中的其他用户见到你的消息，他们就在自己的计算机上保存了一份副本。这会在网络上持续进行。CAT 你的数据的人越多，存储它的人也就越多。

我们的应用程序中到处都需要那个 IPFS 节点，所有的 CRUD/REST 动作都是以它为基础的。在这种情况下，可以考虑创建一个全局变量，那样后续代码写起来肯定容易，但创建全局变量并不是个好主意，因为等规模大了，全局变量会让调试变成一场噩梦。所以我们创建一个 `type`，将变量传到路由调用中的每个方法里：

```
type IPFSHandler struct {
    node  *core.IpfsNode
}
```

我们会用另一个函数把需要的路由器函数封装起来，这样就可以把 `node` 当作参数传进去了。下面看一下向网络中添加数据的代码：

```
func addTexttoIPFS(node *core.IpfsNode) httprouter.Handle
{
return func(w http.ResponseWriter, r *http.Request, ps httprouter.Params)
```

```
{
r.ParseForm()
fmt.Println("input text is:", r.Form["sometext"])
var userInput = r.Form["sometext"]
Key, err := kerala.AddString(node, userInput[0])
if err != nil {
panic(err)
}

}
}
```

我们首先解析表单，然后获取用户输入的文本并赋值给一个字符串变量。接下来用 Kerala 库中的 `AddString` 方法将它添加到 IPFS 网络中。这个方法有两个参数，一个是 `node`，另一个是用户输入的文本，其返回值是一个键值。然后我们输出它。这个键值是刚刚提交的数据的散列值。这样，数据就被添加到网络中了。现在来看看如何从网络中读取数据，并将它们显示在你的个人主页上。

第一次启动这个应用程序时，它会到位于根目录"/"的首页并调用 `TextInput(node)` 方法。跟上一个函数一样，用相应的 `http` 方法封装它，这样就可以将 `node` 作为变量传进去了：

```
func TextInput(node *core.IpfsNode) httprouter.Handle {
  return func(w http.ResponseWriter, r *http.Request, ps httprouter.
Params) {
```

接下来解析 URL，看其中是否有 nodeID（即 peerID）。不管是访问其他用户的主页还是你自己的个人主页，用的都是这个方法。我们会根据 URL 中是否有 `userID` 来做不同的处理：

```
var userID = ps.ByName("name")
```

这会告诉我们 URL 里是否有 `name` 部分。如果没有（说明是你自己的个人主页），Kerala 会用 IPNS 解析策略从你的 nodeID 中拉取 merkleDAG 散列值。如果有，Kerala 会通过解析取得跟这个 `name` 关联的 DAG。因为 DAG 是一个有向无环图，所以每次往上添加散列值，它都会添加到所有之前的散列值后面。这是否说明用户的身份标识一直在变呢？不是的，这就是 IPNS 的好处。Kerala 把 IPNS 和 IPFS 紧密结合在一起。它会将特定 DAG 的 HEAD

节点跟特定的 peerID 关联起来，当有新的数据添加上来时，就重新发布到 IPNS 上。

我们的代码考虑了两种情况。第一种是 URL 中不含 peerID 的，说明要访问你自己的个人主页，应该拉取你的消息：

```
if userID == "" {
pointsTo, err := kerala.GetDAG(node, node.Identity.Pretty())
tweetArray, err := kerala.GetStrings(node, "")
if err != nil {
panic(err)
}
```

在这种情况下，我们从你的 peerID 中解析出 DAG，然后用那个散列值 CAT 你所有的消息。

如果消息数组为空，就给前端返回空：

```
if tweetArray == nil {
fmt.Println("tweetarray is nil")
demoheader := DemoPage{"Decentralized Twitter", "SR", nil, true,
balance }
```

如果有消息，就把这些消息发送给前端：

```
else {
fmt.Println("tweetarray is not nil")
demoheader := DemoPage{"Decentralized Twitter", "SR", tweetArray,
true, balance}
```

另一种情况是 URL 中确实有 peerID。这说明我们要访问其他人的个人主页。

尝试解析那个人的 peerID：

```
pointsTo, err := kerala.GetDAG(node, userID)
```

如果解析成功，要做的事情跟刚才介绍的一样，从 DAG 中取出消息并发送到前端。如果解析失败，说明用户还没有在 Mikro 上发表过消息，所以会返回空。这样前端会显示一个空白的个人主页。

3.2.4　将数据传给前端显示

我们看一下 index.html 中给个人主页模型准备的模板。

首先引入了 Twitter Bootstrap 和 jQuery 这两个非常流行的 Web 前端框架：

```
<link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.
min.css">

<scriptsrc="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/
jquery.min.js">
</script>
<script
src="http://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.
min.js">
</script>
```

引入依赖项后，添加导航栏，然后添加两个主要的 `div`：提交表单 `div` 和消息表 `div`。

```
<center>
<div id="submitform">
<form action="/textsubmitted" method="post">
 <input type="text" name="sometext">
 <input type="submit" value="submittext">
 </form>
</div>
</center>

<br>
<div id="posts">
 <form name="tableForm">
         <body onload="insertTable();">
    <div id="wrapper" align="center"></div>
 </form>
</div>
```

提交表单 `div` 创建了一个标准的文本输入框，并通过 `POST` 提交到 URL `/textsubmitted` 上。当用户点击提交按钮之后，它会带着字符串参数到那个 URL 上，并调用我们之前说过的 `addTexttoIPFS` 方法。

我们准备把消息放到一个 HTML 表格中，以便看起来整齐一些。因为消息是随机添加的，所以表格的大小必须是动态的，并且能在出现新消息时自

动调整大小。我们用 JavaScript 实现这个需求：

```
function insertTable(
{
     var arr = [
            {{range .Tweet}}
                {{.}},
            {{end}}
     ];
          console.log(arr.length);
     var num_cols = 1;
     var width = 100;
          var alignright = "<td style='text-align: right'>"
     var theader = "<table id='table1' width = ' "+ width +"% '>";
     var tbody = "";
     for(var j = 0; j < num_cols; j++)
     {
       theader += "<th text-align='left'><font face='verdana'>My
Posts" + " </font></th>";
     }
          var str1 = "{{ index .Tweet 1}}";
     for(var i = 0; i < arr.length; i++)
     {
          tbody += "<tr>";
             tbody += "<td>";
             tbody += "<b>" + arr[i] + "</b>";
             tbody += "</td>";
          tbody += "</tr>";
          }
     var tfooter = "</table>";
          var endalignright = "</td>"
     document.getElementById('wrapper').innerHTML = alignright +
theader + tbody + tfooter + endalignright ;


}
```

我们用 `{{ }}` 引用通过 `Demoheader` 结构传到前端的数据（消息）：

```
var arr = [
           {{range .Tweet}}
               {{.}},
           {{end}}
    ];
```

然后通过 JavaScript 内建的 `len` 方法获取数组的长度。把这个长度作为 `for` 循环的上限，以便遍历整个数组。在遍历之前，我们先创建一个静态的表

头。然后为数组中的每个对象创建一个新的行对象。在表的每一行中都有一条消息。最后，用下面这行代码把静态元素和循环中创建的动态元素拼接在一起：

```
document.getElementById('wrapper').innerHTML = alignright + theader
+ tbody + tfooter + endalignright ;
```

在文件底部的 `style` 标签中，有一些我添加的样式。它们非常普通，因为这只是个演示程序，用来展示创建一个 dapp 所要做的基本工作，所以界面设计不是重点。

接下来看发现页。主页让你看到自己的消息，而 URL `profile/:name` 让你看到网络上每个人的主页。我们怎么找到这些用户呢？当然是用发现页。大多数社交应用都有类似的发现页，Mikro 也不例外。当你点击导航栏上的 discover 按钮时，会看到类似下面这样的同伴列表：

```
All peers
QmW3ssBgGLANKNKXiRxcQMmxg3FPd3tSwu2Dt96DBLbjBZ
QmRzjtZsTqL1bMdoJDwsC6ZnDX1PW1vTiav1xewHYAPJNT
QmaCpDMGvV2BGHeYERUEnRQAwe3N8SzbUtfsmvsqQLuvuJ
QmepsDPxWtLDuKvEoafkpJxGij4kMax11uTH7WnKqD25Dq
QmUy5jHXui2KzZRC3ofzHKYGmJVqAJTCsRRo2EZ6Wzwee7
```

同伴的标识是 IPFS 生成的默认 peerID。我们来看一下在 app.go 中是如何得到它们的。注意当用户路由到 /discover 时是如何调用 `displayUsers` 方法的。第一步是从 IPFS 中获取所有同伴：

```
//获取同伴
peers := node.Peerstore.Peers()
data := make([]string, len(peers))
for i := range data {
    //假设是小尾数
data[i] = peer.IDB58Encode(peers[i])
}
fmt.Println("the peers are %s", data)
```

我们从 `Peerstore` 中获取所有的同伴，这里内部用的是 IPFS 的 `swarm peers` 命令。接下来创建一个数组，长度跟同伴数量一样，然后遍历它。我们用 `IDB58Encode` 方法将同伴编码成字符串，这样就可以解析它们并把它们都存放在数据数组中。然后将数组传回到前端的 discover.html 页面中。discover.

html 跟用户的个人主页很像，是一个长度会变的动态 HTML 表格，里面是指定同伴的所有消息。唯一的区别在于这个是同伴列表数组，而不是消息数组：

```
var arr = [
{{range .Allpeers}}
{{.}},
{{end}}
     ];
```

3.3 dapp经济学

这部分很有意思。我们把这个小 dapp 转入它自己的小经济体制中。还记得前面讨论的钱币的理想形态吗？彩币是目前在 dapp 内发行资产的最佳方案。你肯定不想为了拥有自己的应用币，就去经受启动一个区块链的痛苦和烦扰。目前，比特币区块链抵抗女巫攻击的计算力等同于 500 多台超级计算机，你没必要从头再来。尽管 Counterparty 提供的解决方案有价值，为这个模型引入了一种新“货币”，但它毫无必要地把事情复杂化了，而且提供的功能也不是模块化的。通过彩币，我们能够在比特币区块链上创建资产。该资产没有所有者，价值随 dapp 本身的价值而变化。

那么如何创建自己的彩币呢？我发现目前最简单的途径是通过 Coinprism（http://www.coinprism.com）网站来做。Coinprism 是一个在线彩币钱包。你可以创建自己的账户，并且将跳转到主钱包页面。创建彩币的费用是 0.0001 BTC。现在这也是没办法的事，只能等待将来有替我们承担这部分费用的服务出现，就像 Onename 替名称币身份标识做的那样。我从 Coinbase 转了 0.0005 BTC 到我的彩币钱包。

接下来，点击导航栏中的 Addresses & Colors，进入页面后点击 + New color 按钮。网站会提示你创建一个地址，如图 3-2 所示。

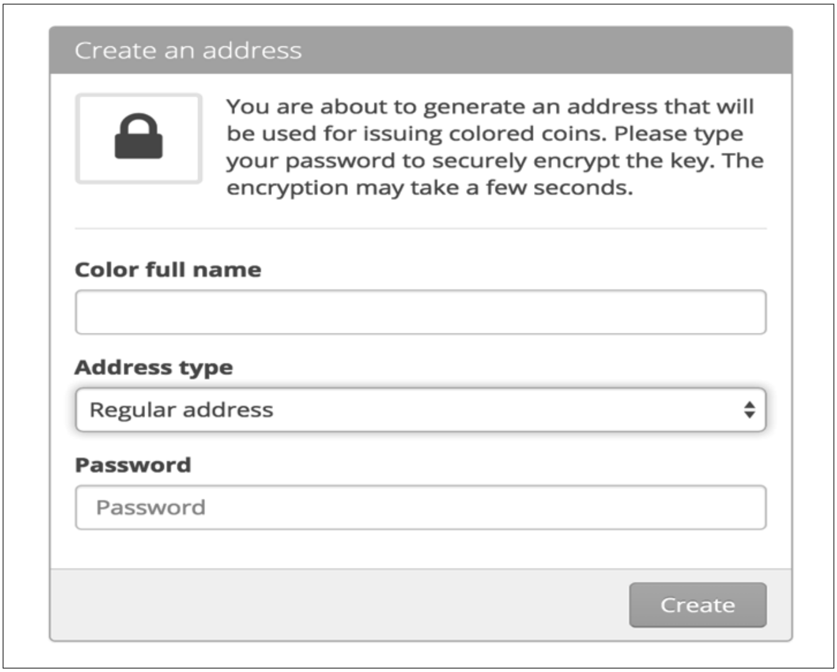


图 3-2：创建一个地址

给你的币种起个名字。它们一般都是以 coin 结尾的，但这不是强制性要求。你的币种甚至可以和你的应用程序用同一个名字。就像 WhatsApp 的竞争对手 Gems 一样。我选择了常规地址，因为不想处理离线存储。

接下来，从你的主地址转一些比特币到新地址，然后就可以用这个新地址发行彩币给你自己了。你的主地址会同时存储比特币和彩币。创建彩币的费用是 0.0001 BTC，并且你可以选择任意数量的份额跟这个彩币关联。我选择了 100 000（如图 3-3 所示），如果你愿意的话可以选择 1 000 000。这个数值最好大一点，这样当你的 dapp 变得特别大的时候才有足够的 dapp 币可用。

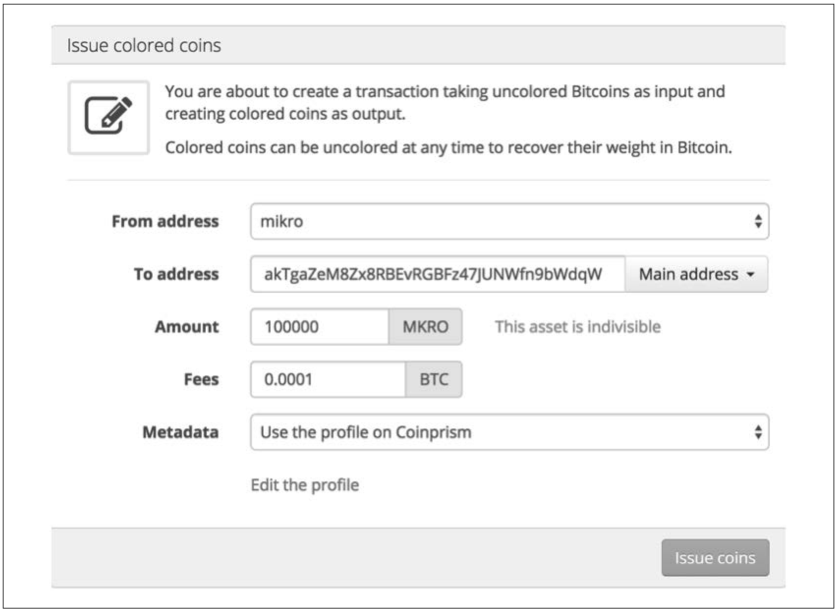


图 3-3：发行彩币

交易完成后，新地址上会有你刚创建的 dapp 币的全部份额，如图 3-4 所示。



图 3-4：你的钱包

完成这一步之后，你应该能在钱包首页上看到这个新币种了，就在比特币额度旁边。恭喜，现在你拥有了刚创建的所有资产。你可以把它们当作你的 dapp 创业公司的股份。你也可以把它们送给任何人。随着创业公司价值的增长，股份的价值也会随之增长。不用通过 IPO 就可以成为一家“上市公司”了。

对于要通过使用你的 dapp 而获利的用户而言，这些资产降低了进入的门槛。它会激励用户扩大网络，以便获取资产访问网络中的稀缺资源。在这个例子中，稀缺资源就是消息。

那么该如何组织 Mikro 的经济体制呢？要在用户发消息时收费吗？可以，但这笔钱给谁呢？最有可能的是付给那些为我们的（加密）数据提供第三方存储方案的节点。但现在因为我们还没有像 IPFS 那样的系统，所以让发消息免费，但查看要收费。也就是说用户可以随便发消息，不会收费，但如果要查看其他用户的消息，就需要先付给作者一小笔费用，其额度是预先设定的。这样用户就会因为向网络中输出数据而收到报酬，然后用赚到的这笔钱查看其他消息，或者用于外部开销。

Kerala 不仅仅是 IPFS 的封装器，还用一个函数调用简化了交易发送。回想一下在所有个人主页上调用的 `TextInput` 方法，如果加载了其他用户发过消息的个人主页，就会调用这个方法：

```
hash := kerala.Pay("1000","1HihKUXo6UEjJzm4DZ9oQFPu2uVc9YK9Wh",
"akSjSW57xhGp86K6JFXXroACfRCw7SPv637", "10",
"AHthB6AQHaSS9VffkfMqTKTxVV43Dgst36",
"L1jftH241t2rhQSTrru9Vd2QumX4VuGsPhVfSPvibc4TYU4aGdaa" )
```

这个函数的作用是付一部分费用给你正在查看的用户个人主页，让你可以看到所有消息，并返回一个交易散列值给你做记录用。

`Pay` 方法的定义和参数是：

```
Pay(fee string, from_address string, to_address string, amount
string, asset_id
  string, private_key string) (string)
```

- `fee string`

  这是需要发送给交易的费用。

- `from_address`

  这是你的资产地址。

- `to_address`

  这是你想阅读的消息所属用户的资产地址。

- `amount string`

  这是你想让用户 A 为了访问用户 B 的消息而支付给用户 B 的金额。

- `asset_id string`

  这是 dapp 的创建者的资产 ID。

现在，所有跟“加密货币”相关的数据都是用 Coinprism API 的测试版处理的。如果你准备好了进入生产环境，可以切换到这个 API 的生产版。参照 Coinprism 上的文档（https://coinprism.docs.apiary.io）来做很简单。（只是在 Kerala 中将测试 URL 换成生产 URL。）

`Pay` 方法会创建一条交易，接收未签名的响应，并计算它的十六进制值。然后对这个十六进制值签名并将其推送到网络中。返回值是这条交易的散列值，你可以在比特币区块链上验证。一切从头开始来发送“加密货币”交易太痛苦了，所以我觉得将它全都抽象成一个方法还是挺有用的。

这里还有一个叫 `GenerateAddress` 的方法。对于你的 dapp 而言，可以设定成在用户第一次运行应用程序的时候调用 `GenerateAddress`，这样他就有了自己的资产地址，可以发送和接收资金。

3.4 遗留问题

这个演示程序中还有些没有实现的功能，但全都是有可能实现的，并且 dapp 很有可能会随着时间进行更新。任何贡献都是欢迎的。

3.4.1 私有网络

你应该注意到了，这个演示 dapp 中没有“朋友”这个概念。你有个人主页，可以发现其他用户，还能在“付费”之后看到他们的消息，但不能像传统的社交网络那样“加好友”。加某人为好友需要数据加密。其基本思

想是你的 DAG 是用公钥 – 私钥对加密过的，在网络上只有那些你信任的节点才能访问你的私钥来解锁和查看你的数据。如果你将某人从好友中移除，应用程序可以生成一个新的公钥 – 私钥对，并将私钥重新广播给仍然是好友的用户。这样，那个被移除的朋友就无法访问你的数据了。这个功能在哪里呢？它叫作 IPFS Keystore，仍在开发之中。等你拿到这本书的时候，它很有可能已经做好了，并且其实就是实现几行额外的代码。你可以在 https://github.com/ipfs/specs/tree/master/keystore 看到它的功能说明。

3.4.2 人类可读的名称

发现页面上的 peerID 看起来不太漂亮。它们是唯一的，但不是人类可读的。前面提到过，名称币从技术上补充了 Zooko 三角，让你可以创建去中心化的、人类可读的、安全的名称。因为 peerID 已经是唯一的了，所以你可以提示用户在名称币的区块链上注册一个人类可读的名称，然后将他们的 peerID 和名称币名称关联起来。只要你查看用户发送的数据，应用程序就能在名称币区块链上验证用户的身份标识，不管用户的 peerID 是否发送了注册交易来请求他们的名称币身份标识。另外一种办法是在应用程序中创建一个信任网络，比如名声系统。当然，最容易的选择是用名称缩短服务（集中式命名空间），但这样会引入一个中心失效点。我个人倾向于选择名称币。

3.4.3 仅显示Mikro上的同伴，而不是IPFS上的全部节点

发现页上显示的不仅仅是使用 Mikro 的同伴，而是 IPFS 网络上的所有同伴。因为 IPNS 很成熟，所以这很容易实现，但我们可以在 IPNS 命名空间中配置一个 app.config 文件。当用户在寻找同伴时，我们可以遍历网络中的每个节点，检查其中是否有 Mikro 特定的签名信息。如果有，就把它列在发现页上。

3.4.4 防篡改支付

在我们的代码中，把用户数据从 IPFS 中拉出来之前，我们通过 Kerala 中的 `Pay()` 支付了一笔。如果有人故意从源码中把 `Pay()` 那段代码删掉，那他在

访问节点数据时会发生什么？这是完全有可能做到的。每个用户可以运行一个监听器，等待在访问他们的资产地址时要发生的那笔费用。如果有支付，客户端才会给支付发送方发送一个能够访问数据的私钥。要得到另一个用户的资产地址，dapp 可以把它作为自己可公开访问的 DAG 中的第一条。

智能合约在所有这些事情里发挥什么作用？名称币有自己内置的智能合约，用于名称注册。自以太坊成立以来，比特币的核心开发人员已经学到了很多教训，比如把脚本语言做得更完整，以便适应更大范围的应用。当然，这只是其中之一。以太坊对区块链技术的研究做了很大贡献，但不幸的是，当涉及去中心化应用程序时，经常并不需要达成全局共识，并且它太昂贵了。

区块链真的非常善于处理金融资产，但计算和存储不行。使用智能合约作为第三方代管服务就是个例子。在交易完成之前，资金一直在区块链上。交易完成时，会收到一个通知，将资金释放到特定的地址上。这个 dapp 不需要用这个，但我们稍后会看到它的用例。

就是这样了！你刚刚用源码构建并运行了第一个 dapp。你可以将其作为起点，去搭建自己的可盈利开源创业项目。

第 4 章

OpenBazaar

本章将会详细介绍去中心化市场 OpenBazaar（https://openbazaar.org）。我们会讨论它所支持交易的基本原理和总体结构，然后介绍 OpenBazaar 的实现，以及它的缺陷和下一步发展的可能性。

4.1 为什么要做OpenBazaar

比特币的出现燃起了人们对下一代电子商务的热情，期待它能带来快速的小额支付和更好的安全性。第一批大规模利用比特币的是 Overstock 和 Dish Network 这样的集中式供应商。比特币给这些主流公司提供了一个机会，让他们得以展示自己精湛的技术实力，但它的匿名性和即时价值转移更适合非法商品市场 Silk Road。

Silk Road 就像是地下版的 eBay。它是一个集中式的网站，但只能通过 Tor 使用**洋葱路由**访问。创建者故意增加了普通用户访问它的难度，它被视为“暗网”的顶峰。人们在 Silk Road 上主要是买卖非法毒品，特别是在那些有严格禁毒法的司法管辖区。尽管公众对 Silk Road 的部分业务感到厌恶，但它上面还有其他商品，比如像大麻这种跨管辖区域销售的软毒品（甚至烟草），或者色情艺术品、图书、珠宝等非毒品。

它运转了很长一段时间，最终还是被美国联邦政府取缔了。这是因为 Silk

Road 有中心失效点：它的所有数据都放在一台服务器上。在政府锁定了那台服务器之后，Silk Road 就被攻下了，所有用户在那个网站上的数据也就都丢失了。另一个用户试图重建这个网站，并将其叫作 Silk Road 2.0；但他最后也被捕了，网站再次被关。

海盗湾（the Pirate Bay）也差不多。这个网站已经被多个政府组织关闭了很多次，但还是不断冒出来。在网站被关之后，网站的所有者就立即决定在哪里再把服务器启动起来，这明显不是长久之计。

除了技术上的漏洞，市场类应用程序的另一个失效点是，其拥有者控制着所有数据的访问。Silk Road 的创始人是 Ross Ulbricht，即“恐怖海盗罗伯茨”。他被拘捕的消息成了世界各地的头条新闻。他目前正在监狱里，因为毒品贩运和计算机犯罪而服刑。在政府解决了 Silk Road 后，人们对去中心化市场的需求变得越来越明显。去中心化市场的访问控制不是由某个人管理的，并且可以运行在任何人的计算机上。正是出于这种需要，OpenBazaar 诞生了。

4.2 什么是OpenBazaar

OpenBazaar 根本没有中心服务器。它是一个政府机构无法限制访问的点对点客户端。OpenBazaar 不是在任何法律许可下运行的，它是自由的全球市场的演化。就像其创建者所说的：“它就像是 eBay 和 BitTorrent 的孩子。”

OpenBazaar 是一个平台，让买家和卖家可以直接相连，不需要把数据存放在第三方来销售商品，也不用支付交易费用。其创建者的理想是创建一个真正自由的交易平台，让人们不需要通过中央权威来发送和接收商品。互联网上从没出现过真正像集贸市场（bazaar）一样的服务：一个真正点对点的市场，买家和卖家可以直接交互，没有任何人在中间观察交易。OpenBazaar 希望把这个概念带到互联网中来。

一个叫“黑市”（DarkMarket）的项目让它的开发人员赢得了多伦多黑客马拉松的冠军，之后改名为 OpenBazaar。如今，这个团队拥有更多开发人员了。他们主要靠接受赞助的方式来获得资金，并没有真正实现盈利。这也是这个 dapp 的主要缺陷：由于对网络成员没有激励，这个商业计划无法扩展。要弥补这个缺陷，可以引入一种元币并让它增值，而不是直接使用比特币。

4.3 OpenBazaar如何运转

OpenBazaar 网络中的每个人都是 P2P 网络中的一个节点。每个人都有三种角色：商家、买家和 / 或公证方。你可以自行选择主要培养自己在哪个角色上的声誉，并且不限于一个角色。目前它用的“货币”是比特币，避开了发行新“货币”这个进入门槛，但这样也使得开发者无法因为自己付出的努力而得到应得的报酬。接下来介绍这三种角色在网络中面对的流程。

4.3.1 商家

虽然 OpenBazaar 的界面仍在开发当中，但网站必需的所有基本元素都已经到位了。商家只需到 setting 标签页为自己的店铺起一个名字，输入档案照片、比特币地址和名称币 ID（可选）即可。在填完这些凭证信息后（见图 4-1），点击保存，这些数据就被保存到他们的本地计算机上了。

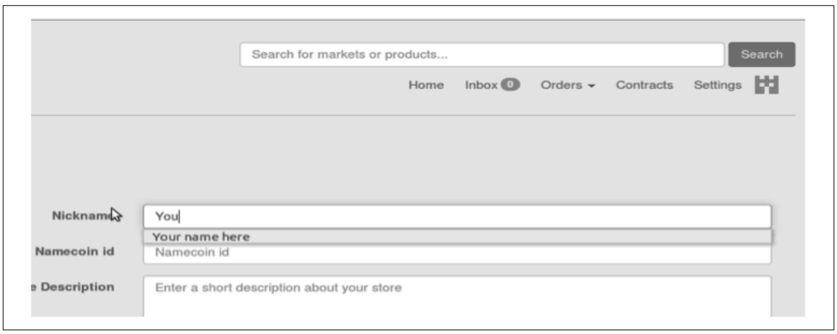


图 4-1：OpenBazaar 凭证信息界面示例

商家还能跟买家沟通：可以使用在 ZeroMQ 基础上构建的消息协议直接在 OpenBazaar 上沟通，也可以通过电子邮件、比特信或他们自己的网站等第三方通信协议进行沟通。因为 OpenBazaar 目前还处于试用阶段，所以更新协议可能会删除商家的店铺数据。开发者为此创建了一个备份选项，让商家可以创建店铺数据的备份，在碰到数据丢失的情况时能够重新融入网络。

当商家在 OpenBazaar 上列出自己的商品后，事情就开始变得有趣了。它用李嘉图合约（Ricardian contract）[1] 的概念来促成网络上的交易。李嘉图合约

注 1：https://gist.github.com/drwasho/a5380544c170bdbbbad8

跟智能合约不同，因为它们不是在区块链上，而是存在于商家的计算机上。这基本上是一种用来追踪交易双方所应承担责任的手段。它表示一个单位的商品。dapp 中使用这些合约来追踪签署协议的双方的责任，并且经过签名的合约是无法伪造的（见图 4-2）。

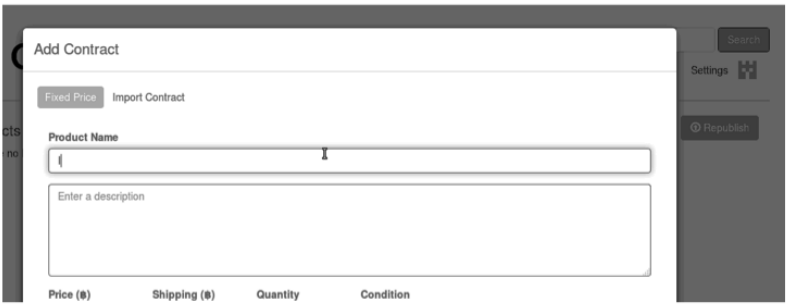


图 4-2：通过用户界面在 OpenBazaar 上添加合约

因此在前端上看，合约只是输入产品和价格的详细信息。此外，它还会把产品链接到卖家的比特币地址和 GUID 上，买家的比特币地址和 GUID 上，以及双方都觉得可以信赖的第三方公证上。

当买家真的下单购买时，卖家会收到待处理订单的通知。买家会收到卖家认可的公证方的详细信息，资金将由公证方保管。如果买家拒绝，公证方会把资金退还给他。如果卖家确实选择相信公证方，就可以将产品发给买家。如果买家收到了商品，他会指示公证方将资金转给卖家。如果买家没有这么做，公证方要负责解决争端。经过对双方信息的处理，他会决定哪一方最有可能说了实话。

4.3.2　买家

买家录入凭证信息的方式跟商家一样，但他们还要选择一个公证方。买家选择公证方，但卖家可以根据公证方的声誉选择接受还是拒绝。在写这本书时，这个 dapp 还处于初期阶段，而且培养信任和声誉需要时间，所以人们最好只做小额交易，以防遇到坏公证。最终，优秀的公证方会升到顶峰，可能会有人将公证服务作为一项业务来做，他们会成为最值得信赖的主导。

4.3.3　公证方

任何人都可以做公证方，只要他们在个人资料中勾选了那一项。只要买家将一方添加为交易合约的公证方，该公证方就可以保管资金、解决争议，以及将资金发送给正确的一方。公证方可以收取解决争议的费用。如果卖家和买家不需要公证方的协助就完成了交易，则无须支付这笔费用。如果需要退资给买家或者从事争议解决工作，公证方将按比例从多重签名中收取费用。公证方的费用会公开展示在其“店面”的服务标签页中。

目前，公证方还是自动接受分配给他们的所有交易，但他们最终将可以筛选交易，并有能力选择接受还是拒绝。在图 4-3 中，我们可以看到其基本功能已经实现了。图 4-4 展示了一个已完成的订单。



图 4-3：OpenBazaar 的公证方界面

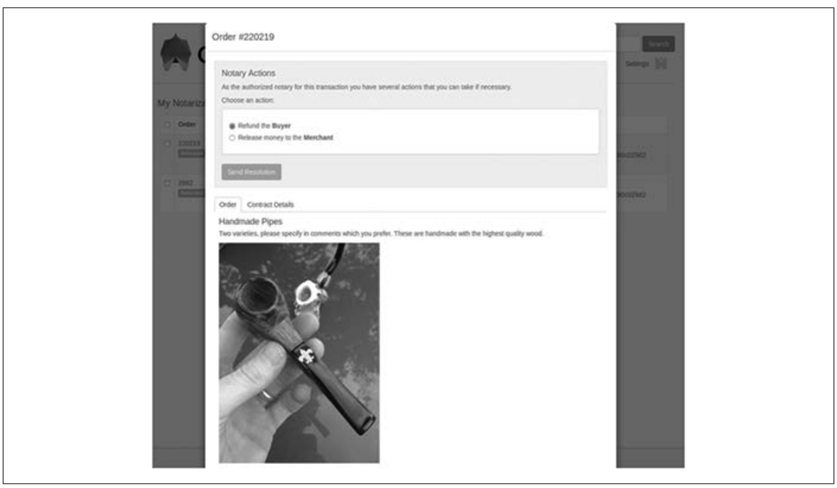


图 4-4：一个手工烟斗的订单示例

4.4　如何安装OpenBazaar

前面已经介绍了 OpenBazaar 是什么，以及它是如何运转的，接下来就下载并试用一下。我们会从技术角度来讨论它，看看开发者在这个 dapp 中选用的技术有哪些优缺点，然后讨论一下他们最终选择用来设计这个 dapp 的方法。

在写这本书时，OpenBazaar 还没有可以直接下载的二进制文件，需要用源码来构建。

首先需要安装 Python。如果你用的是最新版的 OS X，那你的系统中已经装好 Python 2.7 了。否则，需要通过 Homebrew 手动安装。对 OS X 来说，Homebrew 就像 Linux 中的 `apt-get` 一样。在需要从源码编译 dapp 时，Homebrew 超级好用，因为总会遇到至少一个依赖项缺失错误。

要安装 Homebrew，在终端里输入下面的命令：

```
ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install
Homebrew/install/master/in
```

以后就可以用下面这个命令轻松完成上千种包的安装：

```
brew install ________
```

OpenBazaar 是用 Python 开发的。Python 是一种特别棒的面向对象语言，这些年已经积累了很多非常实用的库，有很多分布式项目都是用它做的，比如 RPyc。因此 Python 是个不错的选择。

还需要安装 Python 的模块安装器 Pip：

```
brew install pip
```

现在可以用源码构建 OpenBazaar 了：

```
git clone https://github.com/OpenBazaar/OpenBazaar.git

cd OpenBazaar

./configure.sh

./OpenBazaar start
```

4.4.1 可能会出现的错误

下面是你在自己的机器上构建代码时可能会遇到的错误。记住，这个项目仍在开发活跃期，出错是正常的。

1. 依赖项

既然这个项目还在开发当中，那么你可能会遇到一些依赖项错误。当收到类似于“X wasn’t found.”之类的消息时，不要吃惊，用 Homebrew 手动安装这些依赖项就行了。如果你遇到这样的错误，只要定位到这个依赖项，用 `brew install` 命令安装它，然后再试着运行 `./OpenBazaar start` 即可。如果出现了新的依赖项错误，就重复这一过程，直到把所有依赖项都装上。

2. 端口

你可能会见到下面这些错误：

```
1. 如果你在使用VPN,配置好端口转发或暂时禁用VPN
2. 配置你的路由器,将来自端口62112的TCP和UDP流量转发到你的本地端口
62112
```

这意味着 OpenBazaar 试图使用的一个端口被防火墙或你的路由器封住了。你要检查自己的系统配置和路由器设定，确保这些端口是可以访问的。

3. 数据存储和获取

在 OpenBazaar 中，数据不是存在 DHT 上，而是存在本地的 SQLite 数据库中。我们可以在 datastore.py 中看到 `set_item` 方法，它的输入是一个键值对，包含时间戳和一些凭证信息。它将这个键值对插入数据库中，作为用户计算机本地的记录。请看代码：

```
def set_item(self, key, value, last_published, originally_published,
                 original_publisher_id, market_id=1):

         rows = self.db_connection.select_entries(
            "datastore",
            {"key": key,
             "market_id": market_id}
         )
         if len(rows) == 0:
            self.db_connection.insert_entry(
                "datastore",
                {
```

```
                'key': key,
                'value': value,
                'lastPublished': last_published,
                'originallyPublished': originally_published,
                'originalPublisherID': original_publisher_id,
                'market_id': market_id
             }
        )
        else:
           self.db_connection.update_entries(
                "datastore",
                {
                    'key': key,
                    'value': value,
                    'lastPublished': last_published,
                    'originallyPublished': originally_published,
                    'originalPublisherID': original_publisher_id,
                    'market_id': market_id
                },
                {
                    'key': key,
                    'market_id': market_id
                }
           )
```

然后就可以使用 db_query 方法来查询这些值了：

```
def _db_query(self, key, column_name):

        row = self.db_connection.select_entries("datastore", {"key":
key})

         if len(row) != 0:
            value = row[0][column_name]
            try:
                value = ast.literal_eval(value)
            except Exception:
                pass
            return value
```

只需要一个键来获取必要的值和列名——都是 publisher_id。

OpenBazaar 确实用到了 DHT，但不是用于存储数据。受 Kademlia 的启发（就像 BitTorrent 和 IPFS 一样），OpenBazaar 用 DHT 来存储同伴们的信息，就像“黄页”一样。它是一个去中心化的同伴索引，指导每个节点如何联系其他节点来销售和共享李嘉图合约。当两个节点通过 DHT 相互连接时，

每个节点都能直接从对方那里拉取数据：

```
def __init__(self, market_id, key, call="findNode", callback=None):
        self.key = key
        # 搜索的键
        self.call = call
        # 根据搜索选择findNode或findValue
        self.callback = callback
        # 搜索完成时的回调函数
        self.shortlist = []
        # 要搜索的节点列表
        self.active_probes = []
        #
        self.already_contacted = []
        # 当发送了findXXX后被添加到列表中的节点
        self.previous_closest_node = None
        # 将其更新为搜索过程中发现的最近节点
        self.find_value_result = {}
        # 如果搜索发现了find_value,就是这个值
        self.slow_node_count = [0]
        #
        self.contacted_now = 0
        # 记录联系到了多少个节点的计数器
        self.prev_shortlist_length = 0

        self.log = logging.getLogger(
           '[%s] %s' % (market_id, self.__class__.__name__)
        )

        # 为这个iterative_find请求创建唯一ID(SHA1)以支持并行搜索
        self.find_id = hashlib.sha1(os.urandom(128)).hexdigest()
```

在文件 node/DHT.py 中，类 `DHTSearch` 下的 `init` 方法帮我们搜索 DHT，以寻找那些想要更多数据的节点。它假设你知道节点的键，所以查找很快，但在 OpenBazaar 的 DHT 中，是给每个节点发送广播，所以暴力发现也是可能的。

OpenBazaar 的设计者结构化了所有相关的用户数据，可以作为 JSON 对象发送。这个对象在 protocol.py 中的 `proto_page` 里，叫作 `data`：

```
def proto_page(uri, pubkey, guid, text, signature, nickname,
               PGPPubKey, email, bitmessage, arbiter, notary,
               notary_description, notary_fee, arbiter_description,
               sin, homepage, avatar_url):
    data = {
```

```
            'type': 'page',
            'uri': uri,
            'pubkey': pubkey,
            'senderGUID': guid,
            'text': text,
            'nickname': nickname,
            'PGPPubKey': PGPPubKey,
            'email': email,
            'bitmessage': bitmessage,
            'arbiter': arbiter,
            'notary': notary,
            'notary_description': notary_description,
            'notary_fee': notary_fee,
            'arbiter_description': arbiter_description,
            'sin': sin,
            'homepage': homepage,
            'avatar_url': avatar_url,
            'v': constants.VERSION
        }
        return data
```

用户在 DHT 中被找到并识别彼此之后，就会发送和获取这样的数据。

DHT 的另一个好处就是能用关键字来搜索。因为关键字是用户在店面里定义的，所以可以跟产品或类别关联，让搜索更容易、对用户更友好：

```
def find_listings_by_keyword(self, keyword, listing_filter=None,
callback=None):

        hashvalue = hashlib.new('ripemd160')
        keyword_key = 'keyword-%s' % keyword
        hashvalue.update(keyword_key.encode('utf-8'))
        listing_index_key = hashvalue.hexdigest()

        self.log.info('Finding contracts for keyword: %s', keyword)

        self.iterative_find_value(listing_index_key, callback)
```

OpenBazaar 跟 IPFS 一样用了 DHT，不过它没有像 IPFS 那样实现基于内容地址的数据复制。不管有多少人想要，数据始终放在最初那台计算机上。对于不依赖分布式数据副本的系统来说，没有内置的数据版本管理也是合理的。

4.4.2 身份标识

OpenBazaar 中的节点有自己唯一的 GUID。就像 IPFS 节点有自己的 peerID 一样。在 node/transport.py 中：

```
def _generate_new_keypair(self):

        seed = str(random.randrange(2 ** 256))

        # 移到BIP32 keys m/0/0/0
        wallet = bitcoin.bip32_ckd(bitcoin.bip32_master_key(seed), 0)
        wallet_chain = bitcoin.bip32_ckd(wallet, 0)
        bip32_identity_priv = bitcoin.bip32_ckd(wallet_chain, 0)
        identity_priv = bitcoin.bip32_extract_key(bip32_identity_priv)
        bip32_identity_pub = bitcoin.bip32_privtopub(bip32_identity_
priv)
        identity_pub =
bitcoin.encode_pubkey(bitcoin.bip32_extract_key(bip32_identity_pub),
'hex')

        self.pubkey = identity_pub
        self.secret = identity_priv

        # 生成SIN
        sha_hash = hashlib.sha256()
        sha_hash.update(self.pubkey)
        ripe_hash = hashlib.new('ripemd160')
        ripe_hash.update(sha_hash.digest())

        self.guid = ripe_hash.hexdigest()
```

这些身份标识是通过比特币的 BIP32（等级确定性钱包）协议生成的，通过生成一个新的 SIN 来用 SHA-256 创建你的 GUID。这个 GUID 的唯一性就跟比特币地址的唯一性一样，所以不用担心它会重复。

我们能用比特币身后的椭圆曲线技术给人们唯一的身份标识，就像在 IPFS 中一样，但如何让它们变成人类可读的呢？除了你自己起的昵称，OpenBazaar 还可以输入名称币 ID 作为你的凭证信息。这样，人们基本上可以有两个昵称，能用 GUID 来验证哪个是哪个。这并不是最理想的：也许你可以记住某人 GUID 的最后 5 位和他们的用户名。既然用户可以有一个名称币 ID，这个缺陷还是可以弥补的：

```
def is_valid_Namecoin(Namecoin, guid):
    if not Namecoin or not guid:
        return False

    server = DNSChainServer.Server(constants.DNSCHAIN_SERVER_IP, "")
   _log.info("Looking up Namecoin id: %s", Namecoin)
    try:
       data = server.lookup("id/" + Namecoin)
    except (DNSChainServer.DataNotFound, DNSChainServer.MalformedJSON):
        _log.info('Claimed remote Namecoin id not found: %s', Namecoin)
        return False

    return data.get('OpenBazaar') == guid
```

在这段代码中，每次都会通过 DNSChain 来检查存在名称币地址中的 GUID 是否是那个人所宣称的身份标识，从而判断名称币 ID 是否有效。DNSChain 是一个混合的 DNS 服务器，可以通过 API 轻松地访问名称币数据。

所以 OpenBazaar 的身份标识问题是通过唯一的 GUID 和名称币来解决的，跟 IPFS 类似。

4.4.3 声誉

声誉呢？在任何市场环境中，声誉都非常重要：买家希望能够信任卖家，卖家也希望能够信任买家。在集中式模型中，服务器的拥有者能够把声誉分发给个人，并且只要采取了恰当的安全措施，就不需要应付通过欺骗系统来篡改自己声誉的个人用户。然而在去中心化系统中，检验声誉则要困难得多。

OpenBazaar 中的信任是通过两种协同系统处理的：**全局信任**和**投影信任**。当网络中的所有成员都以同样的方式相信网络中的特定用户时，就是全局信任。这种信任是通过**燃烧证明**和**时间锁证明**建立起来的。投影信任是指向某个节点的信任，对网络中的每个用户可能都不同，所以它是从每个用户投影到节点上的。这种信任是通过一个匿名的部分知识信任网络建立起来的。

下面来看看这些方法的详细情况。

1. 方法1：燃烧证明

卖家必须消耗比特币来创建店铺，这些比特币是不可退回的。所以用户创建多个身份标识的成本很高，这是 OpenBazaar 中抵御女巫攻击的基本方法。尽管这不完美，但还是有威慑作用的。燃烧证明越大，创建账号的成本越高，但潜在用户的进入门槛也变高了。公开并可验证地燃烧一定额度的“货币”是在剩余额度上的**速遣费**（remurrage）。速遣费跟滞期费（demurrage，因为持有“货币”的时间超出给定时段而产生的开支）是相对的。假设你在家里，用笔记本电脑创建了一个有 1000 万额度的“货币”，人们马上开始用它交易。当你出去走了一圈回来后，就只剩下 500 万，其他的都被烧了。如果你持有该“货币”，不管额度多少，就相当于在比特币这样额度固定的“货币”上以常规经济追踪价格拿到了想要的股份。

dapp 首先从节点的 GUID 直接生成一个燃烧地址：

```
def burnaddr_from_guid(guid_hex):
    _log.debug("burnaddr_from_guid: %s", guid_hex)

    prefix = '6f' if TESTNET else '00'
    guid_full_hex = prefix + guid_hex
    _log.debug("GUID address on bitcoin net: %s", guid_full_hex)

    # 扰乱GUID,通过翻转地址的最后一个非校验位来防止SHA256的近似碰撞,
    # 确保不可消费性
    guid_full = guid_full_hex.decode('hex')
    guid_prt = guid_full[:-1] + chr(ord(guid_full[-1]) ^ 1)
    addr_prt = obelisk.bitcoin.EncodeBase58Check(guid_prt)
    _log.debug("Perturbated bitcoin proof-of-burn address: %s",
addr_prt)

    return addr_prt
```

那里只是 GUID 上的一个简单交易。所有节点都可以在 GUID 十六进制上执行同一个 `burnadd_from_guid` 函数来验证某个 GUID 已经烧了“货币”（燃烧证明），并在区块链上验证它烧的额度。

2. 方法2：时间锁证明

燃烧证明让在网络中重新创建身份标识的代价高昂。时间锁证明通过渲染某个时间不可花费的定量“货币”（并将这部分“货币”绑定到一个用户标识上作为“存款”），确保任何时刻都不会有一个真实世界中的实体关联到

大量真实的身份标识。时间锁证明不像燃烧证明那么强大：燃烧证明在效果上相当于永久性的时间锁证明。

在时间锁证明中，节点想要建立对匿名身份标识的信任时，必须以可证明的方式在一个交易内锁定一定量的“货币”，而这些“货币”最终会归还。这个交易有个特点，那就是它在一定时间内是不会执行的。网络知道交易最终会发生，也知道它的额度，以及保持锁定的时长。这些事情都是可以公开验证的。

从心理角度来看，时间锁证明比燃烧证明更容易让人接受。不管怎么说，烧钱会带来心理上的负担，并且可能不太好克服。相对燃烧证明而言，人们很有可能更倾向于使用时间锁证明。

现在的比特币区块链还不允许直接使用时间锁证明机制。尽管比特币协议支持 `nLockTime` 值，但现在运行着的节点还没有兑现这一机制。也就是说，这种交易不会以可公开验证的方式广播。

对以太坊区块链来说，这是个完美的用例，因为它支持图灵完备的智能合约。但 OpenBazaar 还是有意避开了它，主要是考虑到它还没有证明在实践中的可行性，以及在可伸缩性和性能上的各种问题。这是一个明智的决定，侧链提案终将消除这些风险。

3. 方法3：冒险信任（最可行）

开发人员仍在解决信任网络模型的细节问题，以及如何用程序真正实现它，但看起来他们正朝着冒险信任的方向前进。他们一直在酝酿一种想法，想让人们在授信方信任一个当事人的情况下，向其信任的人分配一定的授信额度。所以这个想法基本上就是，如果你真的相信某个人，可以通过一条多方签名的交易给其 0.1 BTC 的授信额度；如果不再信任他了，可以撤回你的授信额度。

信任的指标需要以去中心化的方式永久存储。因为 OpenBazaar 的开发者不想增加比特币上的区块链膨胀（这一直是个有职业素养的好思路），所以转而选择名称币作为替代方案。我发现这真的是个非常合理的方式。

OpenBazaar 的开发者最终可能不会真的实现一个信任网络，因为它甚至不是特别重要。在现实生活中，当有人试图通过诈骗拿走我们的钱财时，我

们只要给银行打电话取消交易就可以收回资金。在 OpenBazaar 网络中，公证方是交易中的关键中间体，并且可能在第一时间阻止诈骗事件的发生。节点只要把信任都托付给他们就可以了，不用信任其他同伴。看着信任如何退出网络很有意思，但我个人还是觉得信任网络是确保网络安全的必选项。

4.5 OpenBazaar还有哪些可以改进之处

OpenBazaar 最重要的缺陷就是没有“内部货币”。比特币提供了直接的流动性，这对于卖家来说很好，但“内部货币”对于早期用户和开发人员来说是个双赢的事情。首先，不管怎么说，第一批使用 OpenBazaar 的用户都要承担一定的风险。因为缺乏声誉良好的公证方，他们的比特币可能会被偷走，而声誉的确立需要时间。如果 OpenBazaar 发行了可以用来在 dapp 内购买商品的自有“货币”，那样会更好。OpenBazaar 可以进行一次山寨币众筹，设定“货币”的初始价格，以及数量有限的代币。

这些“货币”应该是彩币，这样就可以设置一个比特币合约地址（即发送“货币”的地址），来计算向指定的 OpenBazaar 地址发送 OB 币将得到多少回报。随着 OpenBazaar 估值的增长，其“货币”的价值也会增长。OpenBazaar 的早期用户将会因为自己冒险努力促进网络发展而得到回报，买家和卖家的流动性将会增强，最重要的是，这个开源软件的开发人员将会从中得到回报。相对于开源软件来讲，集中式闭源软件的一个主要竞争优势就是资金。后者只是能付钱给顶级开发人员来让他们维护和升级，但有了“内部货币”，就可以把这个模型带到开源软件中来。

另一个有问题的选择是 OpenBazaar 的数据存储模型：只是放在本地的 SQLite 数据库中，没有冗余或数据复制。如果他们用 IPFS 做数据存储，OpenBazaar 的弹性会更好。访问店铺的人越多，店铺的数据副本也会越多。为了让店铺所有者安心，可以告诉他有多少人复制了经过加密的店铺数据。

关于如何构建 OpenBazaar（和构建面对的局限性）及其缺陷和成功之处的知识，对很多应用程序的设计都有借鉴意义。接下来再看看相同的主题在 Lighthouse 中是如何体现的。

第 5 章

Lighthouse

Mike Hearn 是一个比特币核心开发人员，有超过 5 年的经验。他在 BitcoinJ（一个基于 Java 的 Bitcoin SDK）上做了大量的工作，还在世界各地推广介绍比特币，因此赢得了比特币圈的尊重。Lighthouse（https://www.vinumeris.com/lighthouse）是他最新的项目，致力于将众筹去中心化。Hearn 觉得 Kickstarter 和 Indiegogo 这些众筹网站从项目资金中切走的份额太多了，毕竟他们的工作只是维护服务器、发广告、托管和审核提交给他们的项目。再加上 Stripe 和 Amazon 支付这样的服务商收取的手续费，几乎有 10% 的资金到不了募集者手中。Lighthouse 试图去掉中间环节，让募集者能够拿到支持者给他们的所有资金。

此外，Kickstarter 还有地域限制。只有位于北美洲、新西兰和欧洲的人才能在上面创建项目。也就是说，地球上绝大部分地区的人都无法在 Kickstarter 上创建项目。还有一个很现实的问题：司法管辖可以禁用 Kickstarter 这种基于 IP 地址的众筹网站。

除此之外，因为中央权威机构以社区标准的名义给网站定下了规则，所以在 Kickstarter 上是不能创建某些项目的。不是只有集中式资金网站会受到这种限制：尽管存在争议，但通过大多数节点或代理的投票可以在区块链中嵌入所有节点都必须接受的黑名单，从而实现去中心化审核。

另外一个动机是实现无需存款的众筹，因为在去中心化应用程序中存款要冒很大的风险。这样可能会出现很多安全问题，但比特币协议在理论上允

许出现一个可以撤销的中间地带。Lighthouse 可能是第一个实现了协议中这个鲜为人知特性的应用程序。

Lighthouse 也是研究智能合约的好案例。它可能算不上杀手级程序，但因为能帮资金募集者拿到更多钱，所以也可以说是一个很实用的程序。因为比特币比其他支付方式都要轻量，所以 Lighthouse 的支付更加快速。任何资金募集者都可以无障碍进入，支付速度跟协议一样快，不需要银行等中间商的审批。

5.1 功能

检验 Lighthouse 最容易的办法是去它的网站 https://www.vinumeris.com/lighthouse，下载跟你的操作系统相对应的二进制文件，然后双击灯塔图标打开介绍页，如图 5-1 所示。

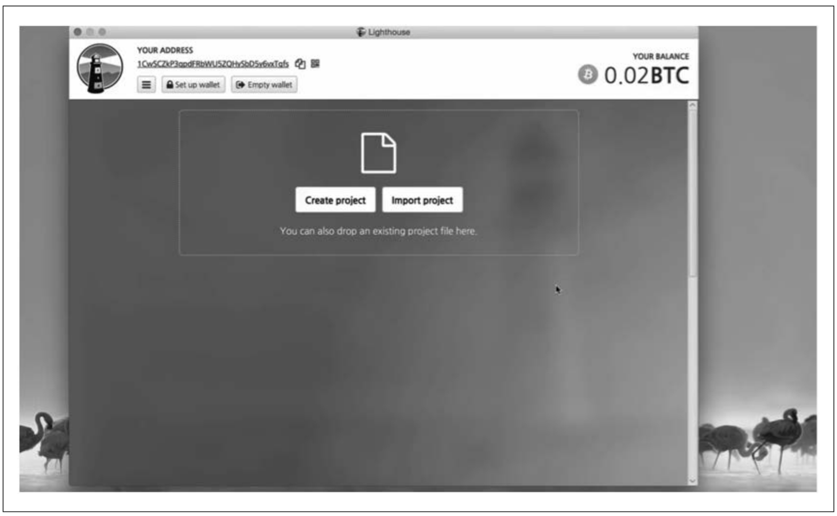


图 5-1：Lighthouse 主页

在介绍页上，你可以选择创建或导入一个项目。它没有发现页（稍后解释为什么）。你可以把已有项目拖曳到页面上，这样会显示它的完整布局格式，然后就可以向它增加资金了（见图 5-2）。

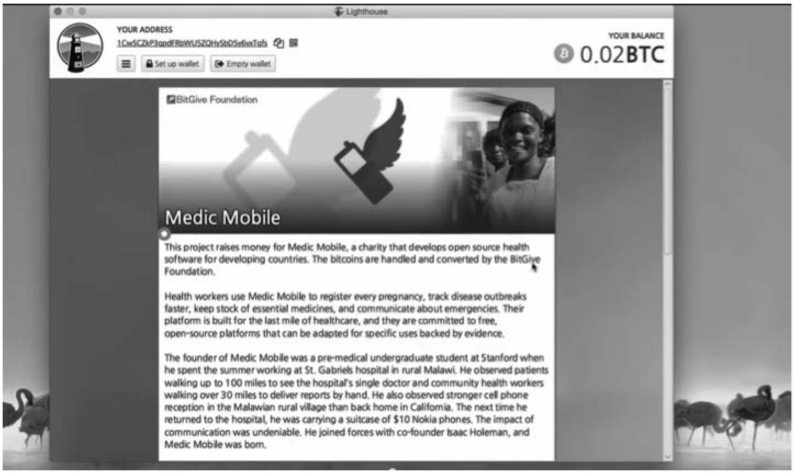


图 5-2：Lighthouse 资金众筹的例子

你也可以向这个项目放入自己的一笔承诺金，如图 5-3 所示。

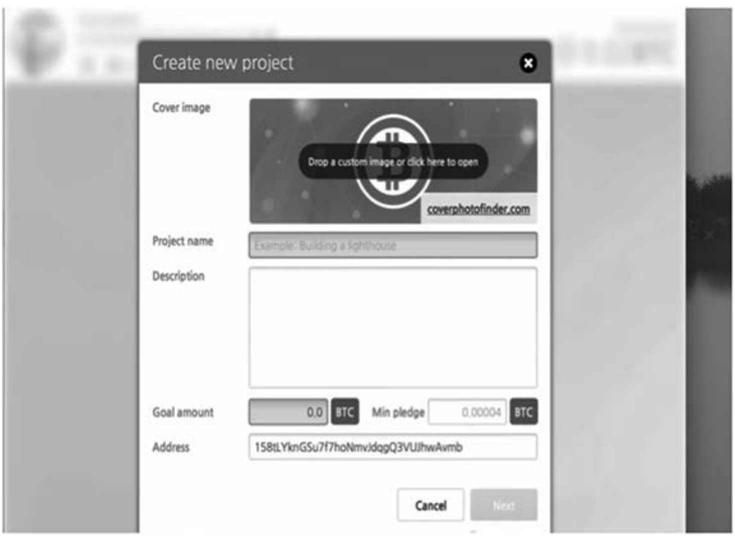


图 5-3：创建新项目

你可以把自己的比特币账号添加到 Lighthouse 上，并用它给项目提供资金。如果项目没有达成目标，创建者没有收集到资金，你随时都可以将资金撤回自己的账户，不用担心受到损失。你也可以开始自己的项目，并在社交媒体上推广它。其他人会下载你的项目文件并提供资金帮助。

你一开始可能会觉得在应用中被迫导入或导出项目文件这种做法不如发现页的效率高，就像第 3 章介绍的 Mikro 程序那样多好。Lighthouse 的创建者之所以没那么做只是因为太难以实现了。他没有足够的金钱和时间来应对由此产生的复杂性和各种问题。去中心化系统开发有难度，并且有一些集中式系统中不会出现的缺陷。跟 UI 同步和状态管理相关的 bug 非常难以调试，比如“我点击了按钮，结果所有认捐都出现了两次。在我重启程序之后，它就好了”。即便不在 P2P 网络中分享数据，Lighthouse 中还是会出现 UI 同步 bug。

如果 Hearn 有先见之明，IPFS 用在这里会非常好。原本可以将 IPFS 作为节点间分享文件的模块。它不像比特币那么昂贵，又有比特币的速度、Git 的版本管理和内容寻址系统的可靠性，每个请求数据的人都会复制一份数据副本。即便不用 IPFS，所有基于 Kademlia 的 DHT 都挺好，然而我们又遇到了去中心化软件开发一直面临的困扰：缺乏资金。一个“内部货币”能帮忙解决这个问题，我们会在本章结尾时讨论。

用户也可以用服务器传输文件。他们可以把文件托管在运行 Lighthouse 节点的联合服务器网络上，也可以将其存储起来。他们可以用自己的个人存储方案，比如 Dropbox 或 Google Drive，然后通过社交媒体把这些文件的链接分享给他人。最近出现了一个叫 Lightlist 的服务，像服务器一样列出了所有的 Lighthouse 项目。这一领域很有可能会出现很多竞争者。这是好事，因为选择多了，不会受单一服务器的控制，意味着更加去中心化。

Lighthouse 有意思的地方是使用了比特币的一个特性。这是一个从比特币 0.1 开始就存在的特性，但似乎完全被忽略了：`SIGHASH_ANYONECANPAY`。Lighthouse 可能是第一个实现这个特性的项目，允许用户用一个注释标记他们的签名，允许其他人参与这项支付。`SIGHASH_ANYONECANPAY` 允许你将多个交易合并进一个大交易中。

当你用自己的私钥给比特币交易签名后，就什么都不能编辑了。所以

把交易广播给网络中的所有人是安全的，反正也没人能编辑你的交易。Lighthouse 的认筹是不完整的比特币交易，会把钱从你的钱包放到募集者的钱包里。因为比特币不允许凭空生钱的交易（只有矿工能凭空生钱），所以要等到每个人都付款后交易才能完成。有了 `SIGHASH_ANYONECANPAY`，如果你得到足够多的认筹，它会把认筹全都合并到一起放进区块链，你最终会得到一个有效的交易。

下面来看一下 Lighthouse 是如何通过程序实现的。我们用 Lighthouse 在 GitHub 上的代码库（https://github.com/vinumeris/lighthouse）作为编程指南。我们不需要克隆它，也不需要用源码构建它。我们知道它能做什么，所以应该深入了解它是如何实现的。

先来看看文件 PledgingWallet.java：

```
public PendingPledge createPledge(Project project, Coin value,
    @Nullable KeyParameter aesKey, LHProtos.PledgeDetails details)
    throws InsufficientMoneyException {
```

这是用来认筹的函数，它将 `SIGHASH_ANYONECANPAY` 操作码的使用和即将要处理的第一个智能合约封在了一起：

```
TransactionOutput stub = findAvailableStub(value);
```

这个方法以有关项目、它的凭证，以及你想要认筹的额度为参数。这段代码试图找到一个能满足给定参数（认筹）的单个输出。

提交多个输入是不友好的，因为会增加认筹者要支付的费用。认筹输出被称为存根（`stub`），使用 `SIGHASH_ANYONECANPAY` 消费它的交易（`tx`）就是认筹。交易模板的输出是合约：

```
Coin totalFees = Coin.ZERO;
Transaction dependency = null;
if (stub == null) {
    final Address stubAddr = currentReceiveKey().toAddress(getParams());
    SendRequest req;
    if (value.equals(getBalance(BalanceType.AVAILABLE_SPENDABLE)))
        req = SendRequest.emptyWallet(stubAddr);
    else
        req = SendRequest.to(stubAddr, value);
    if (params == UnitTestParams.get())
```

```
            req.shuffleOutputs = false;
        req.aesKey = aesKey;
        completeTx(req);
        dependency = req.tx;
        totalFees = req.fee;
        log.info("Created dependency tx {}", dependency.getHash());
         // 这里的变化在于输出在一个随机位置上,所以必须搜索一下
         // 有可能会有两个大小一样的输出,那样的话用哪个都行
        stub = findOutputOfValue(value, dependency.getOutputs());
        if (stub == null) {
            // 创建了一个依赖交易来制作存根,现在找不到它
            // 只有发送了整个余额并为此提取矿工费用后才行
            checkState(req.emptyWallet);
            checkState(dependency.getOutputs().size() == 1);
            stub = dependency.getOutput(0);
        }
    }
```

如果没有那样的输出，这个程序会试图创建一个大小合适的输出再试一次。

然后它会将 `SIGHASH_ANYONECANPAY` OP 添加到交易中，创建一个认筹保证合约：

```
    Transaction pledge = new Transaction(getParams());
    // TODO: 支持在单个认筹交易中的多个输入
    TransactionInput input = pledge.addInput(stub);
    project.getOutputs().forEach(pledge::addOutput);
    ECKey key = input.getOutpoint().getConnectedKey(this);
    checkNotNull(key);
    Script script = stub.getScriptPubKey();
    if (aesKey != null)
        key = key.maybeDecrypt(aesKey);
    TransactionSignature signature = pledge.calculateSignature(0, key,
    script, Transaction.SigHash.ALL, true /* 所有人都可以支付! */);
    if (script.isSentToAddress()) {
        input.setScriptSig(ScriptBuilder.createInputScript(signature, key));
    } else if (script.isSentToRawPubKey()) {
        // 按照现在这个设计,这一分支永远不会执行,因为只能通过一个地址
           才能得到钱
        // 将来可能会支持通过支付协议进行的直接到密钥的支付
        input.setScriptSig(ScriptBuilder.createInputScript(signature));
    }
    input.setScriptSig(ScriptBuilder.createInputScript(signature,  key));
    pledge.setPurpose(Transaction.Purpose.ASSURANCE_CONTRACT_PLEDGE);

    log.info("Paid {} satoshis in fees to create pledge tx {}",
             totalFees, pledge);
```

创建者就是这样用原始的比特币协议实现了智能合约。很丑是不是？原始的比特币协议对开发人员确实不够友好。如果你自己试过，就知道我是什么意思了。因而像 chain.com 这样的服务和封装其丑陋性的 SDK 才会在开发者中间大行其道。

因此，当你做了认筹后，钱其实还在你的钱包里。它只是一个已签名交易的一部分，还不是网络上的有效交易。

当用户决定创建一个项目后，会形成一个 BIP70 支付请求消息。输出很普通，跟常规的支付流程只有若干差别。

- 添加了一个标签为 `title` 的字段。这个字段用几个精简的描述性词语对项目做了总结。
- 添加了一个标签为 `image` 的字段。这个字段包含图片的串行化字节，会被当作项目的个性化内容添加到用户界面中。为了便于重用，Hearn 将这个图片的纵横比设置为跟 Facebook 封面照片一样。
- 如果指定了 `payment_url`，它应该阐明一个允许查询项目状态（当前认筹情况）的扩展协议。
- 支付消息本身必须包含一个只有 `SIGHASH_ANYONECANPAY` 签名的无效交易。只有公开的、已知的、能被认筹消费的 UTXO，才可以当作对项目有效。`memo` 字段也可以包含来自用户的消息（对项目的评论）。还可以出现用于说明详细联系信息的其他字段。

在格式化之后，支付消息要么提交给 `payment_url` 收集并最终跟其他认筹合并，要么用其他方式（比如电子邮件）发送给项目所有者。当项目所有者拿到它之后，可以把这个支付消息加载到自己的 Lighthouse 客户端里。客户端提供了一个 GUI，用于合并认筹并将最终的交易送离 P2P 网络。

随着时间的推移，比特币的脚本语言变得更加强大了，这在很大程度上要感谢区块链技术的创新，以及来自以太坊项目的图灵完备的智能合约。比特币的脚本语言能用于大部分智能合约，但在比特币区块链图灵完备的合约上还有很多工作要做。在 Counterparty 上可以用它，但 Counterparty 有太多非必需的功能，太臃肿了。当然，还有以太坊本身，但侧链代码还没实现，不能让你在比特币上安全地使用以太坊区块链。

要想研究开始陷入比特币协议的政治问题，Lighthouse 是个很好的案例。比特币已经 7 岁了，Mike Hearn 几乎一直都是它的核心开发者。他向比特币核心代码提交过约 44 行代码作为拉取请求，开始被拒绝，然后又被接受了。后来，经过非常激烈的争论，他提交的代码并没有被合并进代码库。即便以他的声望，也没办法对 OP_SIGHASH_ANYONECANPAY 进行简单的修订。

核心开发人员对代码库的防护很严密，这样是对的，因为比特币是非常重要的受托代码，有数十亿美元依赖于它的稳定性。从这个角度来看，它是目前最有价值的独立开源项目。在 Hearn 的 44 行合并请求下，共有来自开发人员的 167 条评论。Hearn 决定创建一个补丁清单来实现“getuxtxo 消息查询 UTXO 集，用来检查认筹”。他甚至说，简化支付验证（simplified payment verification，SPV）在目前阶段是不可能的，因为想改变核心协议需要太多政治活动了。

这样有利有弊：好处是可以防止没有经过认真分析和讨论的改变破坏一切；坏处是会把伟大的变化拒之门外。有望解决这个问题的是，让侧链在保证比特币区块链安全性的前提下引入实验性质的区块链。

Hearn 为 Lighthouse 创建了一个名为比特币 TX 的协议，使用这个补丁集的活跃节点目前大概有 16 个。

5.2 SPV钱包

还记得我们在 Mikro dapp 中用一个第三方 API 创建彩币并在两个地址之间发送它们吗？ Kerala 封装了所有必需的签名和推送。虽然这只是部分的去中心化，但仍然是个不错的起点。更加去中心化的方案明显是在本地运行节点，但在本地运行一个比特币节点要面临的问题是区块链已经增长得太大了。经过 7 年的发展，下载和同步区块链至少要用 4 小时以及很多吉字节的存储空间。替代方案是用轻量的 SPV 钱包来保持去中心化，这是 Hearn 用 BitcoinJ 实现的。

构建一个不验证所有事情的比特币实现是有可能的，但为了保证安全性，要么连接到可信的节点上，要么相信用于校验证明的代理有足够高的难度。BitcoinJ 就是这样的实现。

在 SPV 模式中，客户端可以跟完整节点相连，但只下载区块头部。中本聪在最初的比特币白皮书中描述过这种模式。客户端可以验证区块链的头部是正确连接的，并且难度也足够高。之后，它们向远程节点请求符合特定模式的交易，就像到你的地址的交易。这会通过一个将它们链接到其所在区块的 Merkle 分支来提供那些交易的副本。这个协议让我们用 Merkle 树结构来完成无需整个区块内容的包含证明。

SPV 甚至允许丢弃埋藏得非常深的区块头部来做进一步的优化（可以不存储位于 *X* 头部下面的区块）。如果一个节点是值得信任的，难度就不再重要了。如果你只想随机选取一个节点，那么攻击者挖出一个含有虚假交易的区块序列所要付出的成本应该比欺骗你得到的收益高。通过调整区块的深度，可以用确认时长来换攻击成本。

5.3　身份标识

```
public String signAsOwner(PledgingWallet wallet, String message, @
Nullable KeyParameter aesKey)
{
DeterministicKey realKey =
  wallet.getAuthKeyFromIndexOrPubKey(authKey, authKeyIndex);
if (realKey == null || (aesKey == null && realKey.isEncrypted()))
return null;
return realKey.signMessage(message, aesKey);
}
```

每个项目都有自己的认证密钥。这个认证密钥只是普通的比特币 secp256k1 密钥。它放在用户的钱包里，是从用户的 HD 密钥层级中抽离出来的。项目用它向服务器证明，这个用户就是最初创建项目的人。创建者还应该用这个密钥给消息签名，并在将来证明新版的项目文件是合法的。

作者还创建了一个 BitcoinJ 模板，相当好用。这个模板的 GitHub 地址是 https://github.com/bitcoinj/wallet-template。

图 5-4 是一个设计精良的钱包界面示例。它基本上就是一个用 BitcoinJ 的默认 HTML/CSS 模板写成的 SPV 钱包，你可以在上面创建自己的去中心化应用。它不是特别理想，因为没用元币。尽管它基本上不可能让你赚钱，但这是一个挺好的起点。还有一个叫作 ChromaWallet 的 SPV 彩币钱包，但它

不像 BitcoinJ 一样有个不错的启动模板。把它跟 BitcoinJ 的模板元素结合起来会是一个特别实用的工具，我猜迟早会有人将其创建出来的。

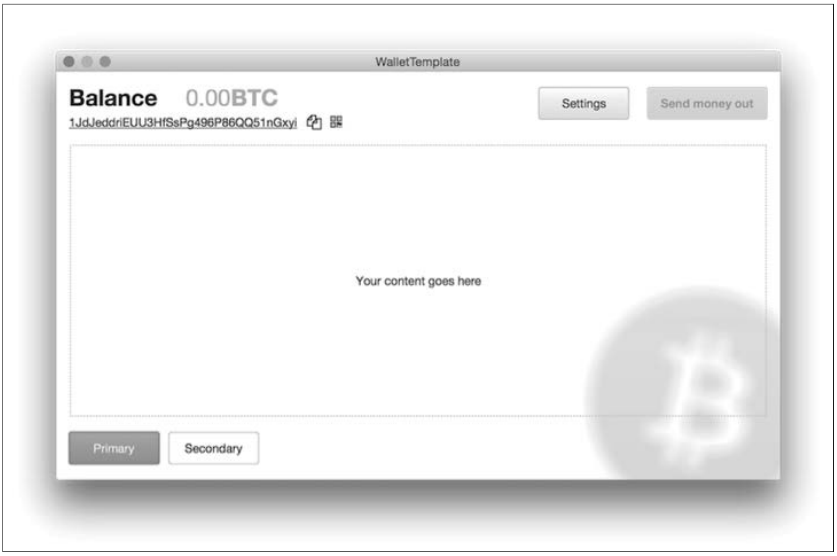


图 5-4：你的钱包余额

第 6 章
La’Zooz

6.1　La’Zooz是什么

在过去几年间，共享租车应用已经席卷全球。其中最大的两家是 Uber 和 Lyft，并且 Uber[1] 似乎要在全世界取得主导地位。仅 2014 年一年，Uber 就赚取了 20 亿美元，是世界上增长最快的初创公司之一。它的前提很简单：利用智能手机已经普及的优势，让用户可以从任何地点叫车前往任何地点。Uber 让任何人都能成为司机，从而将出租车行业的权力去中心化了。此外，借助手机的 GPS（全球定位系统）技术，它让任何人只需按个按钮就可以在任何地方叫车。Uber 出现之后，人们不需要再等出租车了，司机也不用漫无目的地找活儿了。Uber 提供了一个匹配服务，并且很明显把它去中心化了。它是 P2P 技术的最佳体现。

真的是这样吗？最近 Uber 出现了几桩企业文化方面的丑闻。由于其不当的商业活动、激励行为和允许司机做的事情，Uber 已经变得臭名昭著了。高管 Emil Michael 让公司挖掘一个对 Uber 进行严厉批评的女记者以及其他竞争对手的污点。Uber 对其司机施加的权力已经从监管变成了可能是掠夺性的贷款行为。Uber 能实时看到任何地点发生的搭乘数据，以及每个搭乘者

注 1：2016 年 8 月，滴滴出行收购了 Uber 中国。——编者注

的社交数据，这种“上帝模式”已经成为多次争论的主题。与此同时，还有司机刻意呼叫 Lyft 的出租并取消，好让搭乘者转而选择 Uber。

尽管问题不少，Uber 过去几年依然保持了迅猛的增长势头，有数十亿美元的营收。它提供的服务很实用：跟等出租车相比，人们更愿意用位置感知应用来叫车。我们可以很有把握地说，实时的共享租车需求在短期内不会消失。但 Uber 的缺点在于侵害个人隐私，以及这家价值数十亿美元的公司和只能使用其服务的个人之间巨大的权力不对等。

现在人们有了另一个选择，那就是 La’Zooz。搭乘者可以花费 Zooz 代币来搭乘 La’Zooz 司机的车。司机的应用与搭乘者的不同，能让他们在开车时“挖掘”Zooz。La’Zooz 实现了他们称为移动证明的算法。它用 GPS 三角数据追踪司机是否在开车。如果他们在开车，就可以挖掘 Zooz“货币”。

6.1.1 分布式协议

Zooz 代币是如何分配的呢？我们知道给司机的回报就是 Zooz 代币，可以在开车时挖掘。他们挖掘的回报随时间减少，类似于比特币网络。图 6-1 中这个曲线已被证明可以用来激励矿工。

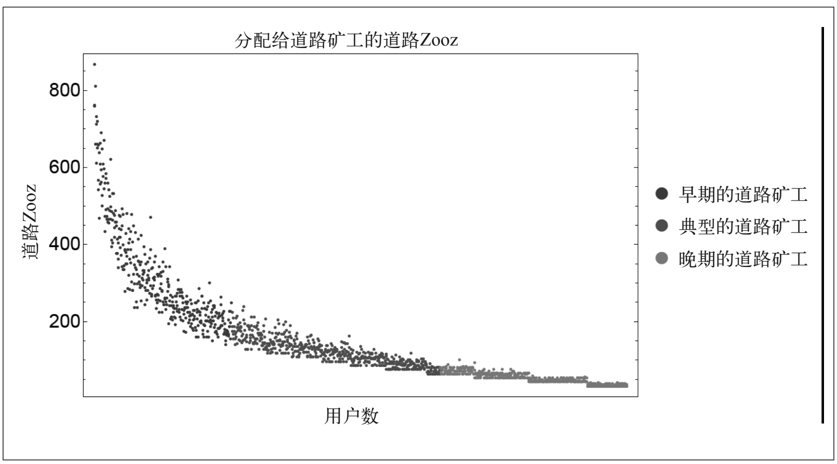


图 6-1：La’Zooz 代币

这个团队创建了一个社区路线图。该路线图是 La’Zooz 项目中未来所有里

程碑的时间线，包括开发、营销和总体增长各方面的目标。他们相信早期用户应该比后来的用户得到更多回报，吸引其他用户加入网络的用户也应该得到回报。

该团队决定为此项目的开发筹集两轮资金。这相当于在通过社交媒体和博客将产品推向公众之前先筹集种子轮来开发一个原型。他们已经建立了一个多重签名的 Coinbase 保险库，需要三个签名中的两个来发放资金。保险库只是一个多重签名比特币地址，任何人都可以向其发送资金。如果三个签名者中的两个同意发放资金，就会把资金发放给发送钱财的人。

多重签名是给众筹买家当作智能合约用的，可以用它发送比特币，并根据他们支付的比例接受相应的 Zooz 代币。可能有下面这三种签名者：

- 一个比特币社区信任的成员
- 一个独立的职业审计师
- 一个 La'Zooz 开发者社区的代表

至于这些人是由谁经过什么流程选出来的，还没有最终决定。在众筹期间售出所有代币的 8% 会被发放给早期购买者作为奖励。为什么是 8% ？为什么在众筹时重新发放而不是直接送给早期参与者？这没有意义，并且令人感到困惑。

如果区域内的司机太多，但没有搭乘者，会有问题；反之亦然。这跟经典的“鸡和蛋”问题一样。La'Zooz 致力于通过算法来检测某个区域内在什么时候有一定数量的司机，从而达到缓解这个问题的目的。如果该区域内司机的数量达到了一个阈值，搭乘者应用会被激活，然后搭乘者才能开始请求搭乘。也就是说，这个应用是一个区域一个区域推出的，就像 Uber 一样，而且完整的部署是由编程而不是手动决定的。

6.1.2　DAO结构

La'Zooz 致力于成为由社区运营的网络，也就是说创建者和用户之间没有差别。每个使用 La'Zooz 的人都属于相同的 DAO（分布式自治组织）。每个月底都会有一次投票，决定每个成员投票的权重和在社区里的回报。每个成员只能给他认识的成员（信任网络）投票来决定其投票权重和收到的股息额度。

新的权重是通过混合计算得出的：75% 的输入来自新的投票，25% 来自上个月的投票。这些数字可以调整，但在我看来，全都分配给新的投票会更好。创建一个 DAO 是种新的体验，成员越容易理解该如何参与，以及你的组织过程是如何工作的，你的 DAO 成功的可能性就越大。在月底的投票开始之前，每个成员都必须写下他们为网络所做的工作。给其他人投票的成员能看到他们自我描述的追踪记录，并判断其意义。

投票永远不会成为一种应尽的义务，这是好事。让成员能选择是否参与投票，而不是承担压力必须去做，免得他们转而去一个更加自由的 DAO。为了在严格和自由之间取得平衡，可以创建社区守则，一开始用集中的方式尽快创建一个初始版本（创建一个团队），然后像社区那样通过协作来增加新的准则。

La'Zooz 继续前进，推出了一本协作白皮书 [2]，介绍了他们的分配机制、分配算法背后的一些数学原理、路线图，以及愿景。自从中本聪在发布比特币时附带了一本白皮书之后，制作 dapp 的人都倾向于随着 dapp 提供白皮书。为了助推这股白皮书热潮，包括 David Johnson 在内的一些知名的风险投资家，真的在宣扬发布一本白皮书是推出 dapp 的“正确”方式。我不这样认为。推出 dapp 的正确方式是提供一份真实的价值主张，一些集中式竞争者无法完成的东西，然后用尽可能简单的方式、大多数人熟悉的方法来解释它。

如果那意味着一个提供信息的着陆页，一个集成了投票功能的社区论坛，以及一个解释你的 dapp 是什么的视频，那就去实现它们吧。白皮书不是必要的。如果把跟 dapp 相关的信息都集中起来放到白皮书里，实际上还可能会造成不必要的困扰。白皮书不需要同时包括商业计划、成员职责，以及跟组织有关的一切事情。

尽管如此，La'Zooz 团队还是继续前进，在以色列创建了一个非营利机构，以确保他们在这个国家是合法的。社区中的每个成员都会得到回报，并且有决定这个 dapp 如何发展的投票权，但是团队本身形成了一个合法的组织。在月底的回报阶段，会有一定比例的股息分给他们，就像其他人一样。

注 2：http://www.lazooz.org/whitepaper.html

有趣的是，按照法律规定，他们必须遵守社区投票的指令来安排自己收到的资金。

这样，DAO 将自己确立为“液态民主”组织。每个人都有投票权，投票是自愿的，并且按权重下放，而创建者仍然得到了代表。组织中复杂问题的大部分都由代表们关照，但如果社区觉得代表们出现了腐败的情况，或者缺乏必要的领导力来维护并发展网络，则随时可以提案投票选出新的代表。

论文中描述的回报机制模糊不清，仍在开发当中。他们应该用股息的形式给付出努力的人们回报。“内部货币”无法在外部流动，只能随着网络的增长获取。我们可以把内部代币当作网络内的股份。可以写一个智能合约，给每个持有 Zooz 代币的公钥－私钥对分配股息。

股息会跟地址持有的代币数量成正比。Dan Larimer 创建了一种名为 bitshares 的山寨币，它用了一种称为代理式权益证明（Delegated Proof of Stake）的共识算法，具有相似的功能。而 Zooz 代币可以被当作 La’Zooz DAO 中的股份和“内部货币”，股息可以是比特币或更多 Zooz。比特币的流动性更好，但 Zooz 的升值潜力更大。这由创建者决定，但我觉得用比特币做股息更好。

6.2 UX

我们来看一些 La’Zooz 创造的设计。到目前为止，去中心化应用程序中还没有出现值得称道的前端界面，但看起来 La’Zooz 了解优秀设计的重要性。

在启动挖矿应用后，用户会看到一条问候消息，并来到一个显示他们的指标的页面。图 6-2 和图 6-3 展示了这个应用测试版中的一些虚拟指标。这个应用是要作为后台进程运行的，挖矿在用户开车的同时进行。在用户挖矿时，甚至可以在前端运行其他应用程序。

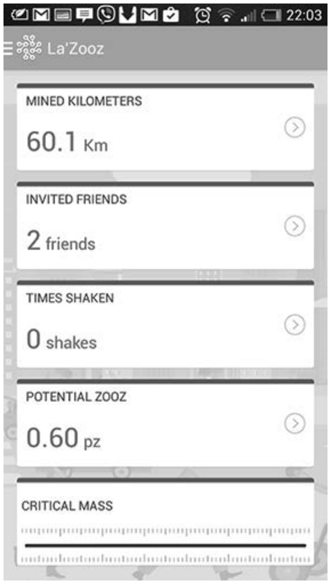


图 6-2：La’Zooz 的状态

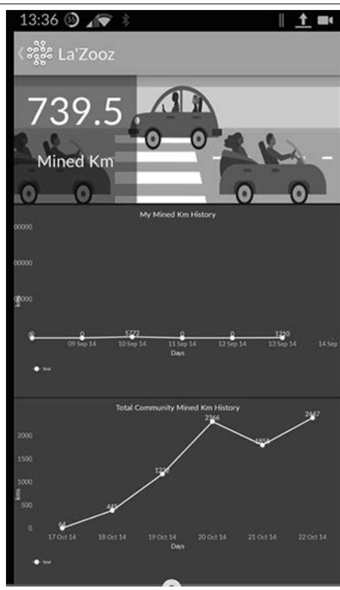


图 6-3：Zooz 代币开采图

点击边栏上的图标，司机能看到自己当前的 Zooz 余额（见图 6-4）。Zooz 应用还是他们的钱包。司机可以用二维码和导出功能与真实生活中的其他人或业务机构互相发送和接受“货币”。潜在的 Zooz 余额是个有趣的标记。如果司机不管到达下一个区块需要多长时间仍然继续挖掘，应用会计算司机能够运行多久。那么阻止女巫攻击的机制是什么？ La’Zooz 如何阻止用户为了赚取更多钱而假装成多个用户来同时运行多个挖矿进程？我们需要到代码库（https://github.com/laZooz/lbm-client）中看一下这个功能。

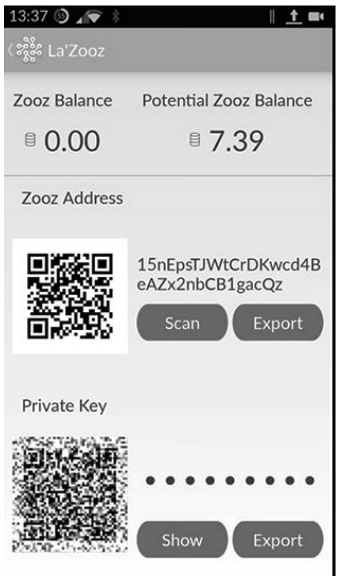


图 6-4：Zooz 余额

6.2.1　架构

1. 数据存储和获取

我们在 Mikro dapp 中用 DHT 存储数据，用 BitTorrent 传输协议来获取数据。这全都要归功于 IPFS，它是前述两项技术的集大成者。OpenBazaar 中也用到了 DHT，但没有内置数据复制功能，所以不够健壮。如果节点下线了，并且之前没有人看过其数据，那么节点中的那些数据也跟着下线了。

在 Lighthouse 中，开发人员甚至没有尝试使用去中心化数据存储，因为那太难实现了。他们选择在网络中广播项目，由参与者共享并下载项目文件后加载到自己的 Lighthouse 实例中。那么 La'Zooz 是如何处理数据的呢？我们在源码中可以看到一个名为 ServerComs 的文件。服务器通信？听起来不太像是去中心化的。我们看一下这个类中的三个方法：

```
public void registerToServer(String cellphone)
{
    String url = StaticParms.BASE_SERVER_URL + "api_register";

    List<NameValuePair> params = new ArrayList<NameValuePair>();
    params.add(new BasicNameValuePair("cellphone", cellphone ));
    params.add(new BasicNameValuePair("accountname", accountname));
    this.postRequestToServer(-1, -1, url, params);

}

public void setLocation1dddsfsdfs(String UserId, String UserSecret,
    String data)
{
    String url = StaticParms.BASE_SERVER_URL + "api_set_location";

    List<NameValuePair> params = new ArrayList<NameValuePair>();
    params.add(new BasicNameValuePair("user_id", UserId ));
    params.add(new BasicNameValuePair("user_secret", UserSecret ));
    params.add(new BasicNameValuePair("location_list", data ));

    this.postRequestToServer(-1, -1, url, params);
}

public void getUserKeyData(String UserId, String UserSecret)
{
    String url = StaticParms.BASE_SERVER_URL + "api_get_user_key_
data";
    List<NameValuePair> params = new ArrayList<NameValuePair>();
    params.add(new BasicNameValuePair("user_id", UserId ));
    params.add(new BasicNameValuePair("user_secret", UserSecret ));

    this.postRequestToServer(-1, -1, url, params);
}
```

那么，看起来这三个方法分别在用服务器获取用户数据，设定用户位置，并为用户注册账号。那个 `BASE_SERVER_URL` 变量是什么？在 `StaticParams` 类中，我们找到了它的定义：

```
public static final String BASE_SERVER_URL = "https://client.laZooz.
org/";
```

我们发现，它把数据存储在一个中心服务器上，并从中获取数据。作为一个研究过 dapp 的人，我并不是特别吃惊。这个领域中的噪声太多了——很多项目真的是宣传得很厉害，有很多炒作、很多追随者，也做了很多承诺，但最终却在数据存储这样关键的地方投机取巧。这可能是因为找不到像 IPFS 这样优秀的去中心化存储工具，也可能仅仅是因为不知道怎么做 dapp 来保证充分的去中心化。把用户数据存在服务器上使得这个应用跟 Uber 差不多，只是它用了“内部货币”，并且有一个协作式的组织机构，而不是一家公司。La’Zooz 其实可以实现一个 IPFS 的 Android 封装，地址是 https://github.com/dylanPowers/ipfs-android。

2. 山寨币

La’Zooz 用什么区块链来发行它称为 Zooz 的“内部货币”？

```
protected String doInBackground(String... params) {

ServerCom bServerCom = new ServerCom(MainActivity.this);

JSONObject jsonReturnObj=null;
            try {
                MySharedPreferences msp = MySharedPreferences.
getInstance();
                bServerCom.getUserKeyData(msp.getUserId(MainActivity.
this), msp.getUserSecret(MainActivity.this));
                jsonReturnObj = bServerCom.getReturnObject();
            } catch (Exception e1) {
                    e1.printStackTrace();
            }

String serverMessage = "";
            try {
                if (jsonReturnObj == null)
                    serverMessage = "ConnectionError";
                else {
                    serverMessage = jsonReturnObj.getString("message");
                    if (serverMessage.equals("success")){
                        String ZoozBalance =
jsonReturnObj.getString("Zooz_balance");
```

这个方法在跟服务器请求钱包余额。这意味着钱包不是放在本地，而是放在服务器上的。又是集中式的！它不是第三方钱包托管，而是 La'Zooz 自己的服务器。接着看他们用的区块链类型。区块链在他们的服务器上，我们得不到客户端代码进行研究，但从他们在各种社交媒体上的发言来看，我敢肯定他们用的是 Mastercoin 区块链。

Mastercoin 是在比特币区块链之上的一层。它通过交易往区块链里插入数据，从比特币矿工的角度来看，这些数据是没有意义的。Mastercoin 不像比特币或山寨币那样依赖自己的区块链，它无法充当智能合约引擎。任何人都可以从给定地址进行 Mastercoin 的双重消费。什么都阻挡不了有人在区块链中发布冲突的 Mastercoin 交易。Mastercoin 协议做的唯一一件事情是定义了一条规则，用来判断应该忽略哪条交易。

不仅如此。La'Zooz 的一些功能要求用户积极参与，但协议中没有要求他们行为正确的任何约束。比如说，Mastercoin 协议有个功能叫"注册数据流"，比特币地址的所有者可以宣称他们会发布隐藏在交易中的数据。所有者可以承诺每周将汽油的价格发布到数据流中，但实际上没什么能要求他定期提交数据。更糟糕的是，没什么能阻止他撒谎。这个缺陷使得整个数据流对智能合约来说是无价值的输入。

跟 Mastercoin 不同，彩币在区块链中插入的数据非常少，所以对于矿工来说非常轻量。智能合约可以用比特币的内部脚本系统创建，也可以用采用了图灵完备的合约创造器的侧链创建。

6.2.2 合约

这样，我们发现数据集中在他们的服务器上，而钱包节点是连接到 Mastercoin 网络上的。为了实现自动众筹，应该允许使用智能合约，并且他们说使用了一个多重签名保险库来释放资金。但怎么支付股息呢？自动托管支付是智能合约的标志。合约应该在区块链中。看到他们是如何用 Mastercoin 协议使用比特币区块链的，很明显必须用比特币脚本语言制作智能合约。看一看以太坊的主页，会发现 La'Zooz 是为了智能合约而使用以太坊的项目之一。

以太坊很棒，他们用于构建智能合约的工具比比特币的成熟，但问题是其

区块链还没有得到证明，并且可能会大幅缩减尺寸，否则就得一直在安全问题上玩“打地鼠”游戏了，就像 Gavin Andresen（比特币的主要开发人员）说的那样。最好的情况是，当侧链协议发布之后，以太坊会成为比特币的侧链，那时候有需要的人不用再冒险依赖以太坊区块链的安全，就可以编写图灵完备的合约。如果你想用你的 dapp 赚钱，使用以太坊区块链就是不明智的做法。

6.2.3 改善

La’Zooz 是一家雄心勃勃的创业公司。他们要创建第一家这样的 DAO：不仅涉及分散的劳动力，而且这些劳动力接受股息、有投票权，并跟现有的法律架构相一致。在开发阶段，它可以用集中式的数据、财富和身份标识作为替代。就像在眼下，尽管它的初衷看起来非常好，但似乎冒险引入了好几个失效点。

La’Zooz 应该用彩币协议发行资产。在网络中既应该把 Zooz 代币当作股份，也应该把它当作“货币”。还要创建一个智能合约，根据每个人在网络中的权益按比例给他们发送股息。真正的 Zooz 应用程序可以是彩币的 SPV 钱包，是完全去中心化的，但如果想让开发简单一点，可以用 Coinprism。

La’Zooz 组织 DAO 的方式值得称道。如果有一份法律合同要求那些在以色列这家公司的人都遵循社区投票的结果，那么它就遵守了地方法规，又通过“液态民主”坚守了去中心化原则，从而避开了 DAO 的法律空白地带。应用中应该有投票功能，就像侧边栏上有标签一样。

用户应该选择投票。任何人都可以提交提案来请求某个功能、集中营销的新区域或者更换领导。所有人都应该可以对这些提案投票。投票论坛会按照 Reddit 的风格工作，得票最多的会升到最上面。法律确定的 DAO 领导或代表会按照法律的要求通过这些提案。

数据不应该集中，但是别无选择。应该用一个像 IPFS 那样使用 DHT 的数据存储和获取服务。用团队自有的服务器存放所有数据不符合去中心化的目的，不过 La’Zooz 的情况稍有不同，因为它在法律上只做社区想要的事。社区最想要完全的透明。那种透明意味着如果没有社区的同意，没人能把他们的数据卖给第三方。这么做是违法的。

La’Zooz DAO 中的工作或者是按角色指定的，或者是全栈的。初创行业已经涌现出了全栈专家，但就 DAO 而言，全栈不仅意味着要做工程上的工作，而且要胜任公司里各种角色的工作，包括营销、工程和客户关系。用去中心化的方式招聘、评审和解雇员工会比较困难。进入 DAO 应该没有障碍，评审过程正如 La’Zooz 所说（对成员回报的投票和投票的权重）。解雇的提案可能是感到不满的员工提出的。

解雇基本上就是以一种分布式的方式禁止某人的活动。坏人可能会对网络进行 DDoS 攻击，或者上传儿童色情文件。人们应该投票解雇某个成员（那些认识这个成员的人），然后 DAO 的代表会实现一个存储在所有节点本地的黑名单来封禁那个人的地址。也就是说，如果某个地址出现在了黑名单上，这个人就不能跟网络上的其他节点进行交易了。黑名单会存在区块链上，以便所有节点都能达成共识。

La’Zooz 不应该在侧链提案还没准备好时就为了智能合约使用以太坊。尽管比特币脚本语言不是完全图灵完备的，但还是能处理托管自动付款的大部分情况。

最后，La’Zooz 不应该把 DAO 的所有指南都放到一份白皮书中，而是应该把信息模块化成很多部分，以便于懒人阅读。还应该让信息便于查阅，因为网页比 PDF 更容易搜索。

6.3 总结

对这几个 dapp 的介绍应该能激发你的一些想法，让你动手开发自己的 dapp。在**开放性**和**去中心化**的双重指导下，你不会出错的。

关于作者

Siraj Raval 是一名 dapp 开发人员、企业家，从心底里喜欢讲述技术故事。他是一位全职的 YouTube 明星，有自己的节目 *Sirajology*。他是开发人员众筹平台 Havi 的创始人，开发过包括 Meetup 在内的几个 iOS 应用，还做了很多开源方面的工作。除了程序员，Siraj 还是一位旅行家、音乐家、后现代主义者和潜水员。

关于封面

本书封面上的动物是红胸棘鲷，学名是地中海胸燧鲷。

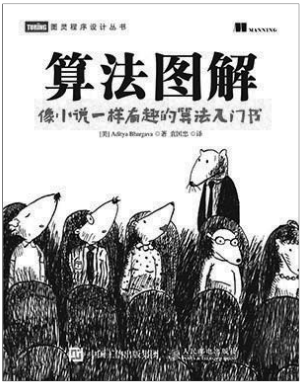
这种深海鱼类广泛分布于大西洋和西印度洋 100~1175 米深的水域内。红胸棘鲷是一种小型鱼，只能长到 42 厘米。它有椭圆形的外形、大眼睛，以及分叉的尾巴。

目前已知红胸棘鲷能活 11 年，主要吃甲壳纲动物。

O'Reilly 封面上的动物大多数是濒危物种。对我们的世界来说，它们全都很重要。要了解如何提供帮助，请访问 animals.oreilly.com。

封面图片来自"多佛画报档案"。

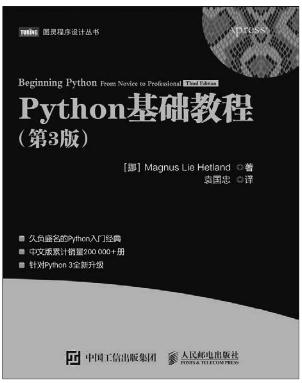
技术改变世界 · 阅读塑造人生



算法图解

- ◆ 你一定能看懂的算法基础书
- ◆ 代码示例基于Python
- ◆ 400多个示意图，生动介绍算法执行过程
- ◆ 展示不同算法在性能方面的优缺点
- ◆ 教会你用常见算法解决每天面临的实际编程问题

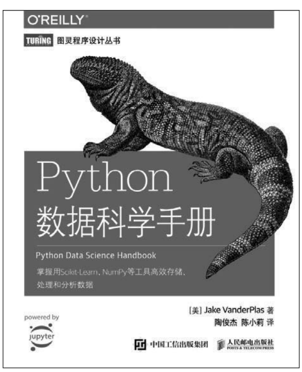
作者： Aditya Bhargava
译者： 袁国忠



Python 基础教程（第 3 版）

- ◆ 久负盛名的Python入门经典
- ◆ 中文版累计销量200 000+册
- ◆ 针对Python 3全新升级

作者： Magnus Lie Hetland
译者： 袁国忠



Python 数据科学手册

- ◆ 掌握用Scikit-Learn、NumPy等工具高效存储、处理和分析数据
- ◆ 大量示例+逐步讲解+举一反三，从计算环境配置到机器学习实战，切实解决工作痛点

作者： Jake VanderPlas
译者： 陶俊杰　陈小莉

微信连接

回复“区块链”查看相关书单

微博连接

关注@图灵教育 每日分享IT好书

QQ连接

图灵读者官方群I：218139230
图灵读者官方群II：164939616

图灵社区
iTuring.cn

在线出版，电子书，《码农》杂志，图灵访谈

O'REILLY®

去中心化应用：区块链技术概述

随着区块链技术的发展，传统的服务器－客户端模型可能会被颠覆，开发人员不用去购买云服务器部署自己的应用。有了区块链这个基于全球服务器、全天候不间断服务的应用平台，开发人员可以轻松地把应用部署到区块链上。此外，由于"代币"机制的存在，开发人员可以很容易地获得价值回报。相信在不久的将来，dapp会超越传统app，在人们的社会生活中扮演越来越重要的角色。

"凭借优越的去中心化架构、灵活性、透明度、弹性以及分布式特性，去中心化应用终将在公共事业、用户基础和网络估值方面超过各大软件公司。"

——David A. Johnston

公证通（Factom）公司董事长

内容特色

- **了解使去中心化数据、财富、身份标识、计算和带宽成为可能的分布式系统技术有何优势**
- **利用Go语言、去中心化架构、去中心化消息应用和点对点数据存储构建"另一个Twitter"**
- **理解OpenBazaar的去中心化市场及其支持交易的结构**
- **探索去中心化众筹项目Lighthouse如何超越Kickstarter和Indiegogo等对手**
- **深入讨论直接连接乘客和司机的P2P共享租车应用La'Zooz**

Siraj Raval

dapp开发人员、企业家。他是开发人员众筹平台Havi的创始人，开发过包括Meetup在内的几个iOS应用，还做了很多开源方面的工作。除了程序员，Siraj还是一位旅行家、音乐家、后现代主义者和潜水员。

COMPUTERS / FINANCE

封面设计：Randy Comer 张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机／互联网金融

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-47930-3

9 787115 479303 >

ISBN 978-7-115-47930-3
定价：39.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks